TU WIEN Informatics

Hybrid methods for the Bus Driver Scheduling Problem

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dott.mag. Tommaso Mannelli Mazzoli
Registration Number 12046158

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Professor Dr. Nysret Musliu

The dissertation has been reviewed by:

Maria Teresa Ortuño	Andrea Schaerf

Vienna, June 26, 2025

Tommaso Mannelli Mazzoli

Erklärung zur Verfassung der Arbeit

Dott.mag. Tommaso Mannelli Mazzoli

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 26. Juni 2025

Tommaso Mannelli Mazzoli

Acknowledgements

It's half past six, and I'm resting on the couch, waiting for the dance class. Misko sat close to me, and I asked him, "Why do you do yoga every day?" He replied,"It makes my life easier" 🦁. That simple statement made me wonder: "**what made my PhD easier?**". I hope to answer that question in the following pages.

The person I am most grateful to is my mother, Monica. She made so many sacrifices for me that I will never fully repay her for everything I have received. She has dedicated her life to my well-being and keeps taking care of me. *Grazie mamma.* Thank you also to my *famiglia*: Anna, Carlo, Gaia, Lapo, Michela, for being close to me all this time.

I want to thank Eleonora Nesterini. Thank you for being there. Thank you for being the "little star I bring in the moments when I have no light". Thank you for kindly listening to my cries and my angry words without judgment. Your unique sweetness in empathising is splendid, and I am grateful to have experienced it. You are special, and this document belongs to you too. Indeed, this document wouldn't exist without you. I am grateful to have you in my life. You made my life much easier.

My *guru* is without a doubt Fulvio Gesmundo, who I consider the most intelligent person that I have ever met in my life[1]. I am grateful to have you in my life and consider you a great friend.

In Vienna My supervisor, Nysret Musliu, is a great and kind man. He can calm me down and solve the most difficult problems that seem unsolvable to me. I am honoured to have met him and to have been one of his PhD students. Thanks to his benevolence, I felt respected and appreciated. We discussed everything, from the best Asian restaurant in town to the most complicated geopolitical issues. I still remember several times he found me at the office overworking, and he told me to go home and not to work anymore, saying that it was good enough. This is precious advice and often not valued enough. Sometimes, I even thought that our relationship was more like that of a father and son. You made my PhD easier, *faleminderit shumë.*

My moral cosupervisor is Lucas Kletzander, who helped me almost every week deep into the most tricky bit of Python code. Lucas, you made my PhD so much easier. *Herzilchen Dank*

[1] We still have to break the tie with Pietro tho🤣

Thanks, Ida and Tobias, for sharing most of our working lunch with me every day for the last three years. You showed incredible patience with me, and I appreciate it. You made my life easier[2].

I also want to thank the people at the office who made this journey more bearable: Aleksandar, Anton, Anouk, Davide ML, Davide S, Ennio, Esther, Federica, Francesco, Friedrich, Giovanni, Ignacio, Mark, Marton, Michele, Nelson, Oliviero, Petra, Sarah, Sanja, and Shqiponja.

Thanks to my colleagues Felix (and Eni), Florian, Francesca, Luca, Marc, Mimi, Patrick, and Roberto for our nice time together.

A very special thanks to Markus Hecher and Bettina for helping me survive the last days of December 2023, literally. *Vielen Dank*.

Many other people tagged along during this journey on various occasions. Thanks to Altana, Ambika, Ferdinando, Jelena, Khaled, Margarita, and Orion. Thanks also to Tatyana for teaching me the German language.

Down under in Australia From March to June 2023, I had the fantastic opportunity to go to Melbourne, Australia, for three months. First, thank you, Kate Smith-Miles, for being my host, both at your office and at your place. I will keep the precious video where I play the piano with your husband. And your garlic bread was so delicious!

Over there, I met amazing people who contributed to making me love that stunning place. I want to thank Hritika for introducing me to the campus and sharing so much in so little time. Then, thank you, Jee, for the fun time playing table tennis, joking about tiger parents and the quirky night experiences in CBD. I am grateful to Eliana for her kindness and our *chismes*.

Thank you, Gianni, for inviting me to the Karate class at World Shotokan Karate-DO in Coburg. It was a fascinating experience, thanks to sensei John Haitidis's great words.

Thank you, Alessia, Alysson (thanks for the T-shirt, I still have it!), Alberto, Charlotte, Connor, Eric, Franklyn, Giulio, Harry, Henk, Jip, Mazzy, Vera, Yi.

Thank you Prof. Jen Martin for the splendid presentation about Science communication. It was a very precious resource, like your podcast Let's Talk SciComm. Also, thank you for having told me about Randy Olson and the strategy of "And, But, Therefore", rather than the boring (albeit common) "And And And".

Thanks to Laura and her friends Esteban and Olivia. The karaoke night at the student hotel was so much fun!

Barcelona From September to December 2023, I had the opportunity to do a *second* research stay. Thank you, Christian Blum, for hosting me, working with me with

[2]Except when we had to decide where to have lunch and the several initial options narrowed down to that tasteless and bland place that I don't want to mention.

enthusiasm, and having the enormous patience to tolerate my personal and frequent questions. Thanks also to Pedro Pablo Pinacho-Davidson for collaborating with me [3]

The IIIA is an amazing place to do a research stay at. Thanks to people like Athina, Björn, Camilo, Elifnaz, Israel, Jairo, Jaume, Mehmet, Rocco, Shuxian, and Stephanie, I enjoyed my time there, it was great fun. At IA, everyone was so sweet to me; even the keeper who used to kick me out of the building at 20:00 did it with care and a smile.

Moltes gràcies to my flatmate Mireia, who was my mentor there and who taught me many paramount things. I still remember when we were meditating, and she asked me "How do you know you have a hand without touching or moving it?". Deep.

Thank you, Claudia, for the nice time we spent together.

Dancing I believe that dancing was a game-changing event in my life for three reasons:

1. Changing environment and people help disconnect from the PhD problems
2. You start seeing your body in a different way and tolerating more and more the image you see in the mirror
3. You meet many people who you share a common passion with

My first experience with bachata was at Miss Collins, a club in Melbourne that was organising free bachata classes. I want to thank them because their free class made me discover a new world, a new Tommaso. Thank you, Deana and Yohann and the Bachata BEATS staff: Blagoja, Ivan, Laura, and Rina.

While in Barcelona, I spent most of my free time at the Sant Cugat Academy, where I started dancing salsa and improved my bachata skills. Thanks, Dani, Eudys, Mei, and all the smiling fun people I met there.

When I returned to Vienna, I kept dancing at Mydance dance school. I am deeply grateful to Sergio Mendoza Matos and Zsuzsanna "Beige" Böjte for teaching me the basics and the joy of dancing Sensual bachata and Cuban salsa with incredible professionalism, passion, and enthusiasm. Thank you to the other great teachers: Ada, Alex, Dani, Deedee, and Sandra. I appreciate the kindness of Amal, Iris, Lucia, Olena, Ricardo, and Shuntaro. Thanks to all the people who I met there. Frankly, I don't even remember most of their names (sometimes you never ask, sometimes you forget). However, they made my life easier. Special thanks to Caterina Carbone: she is such a fun and funny person, and I really enjoy talking to her. I am glad to have met you, Cate!

I am grateful to the owner of the (unfortunately now torn down) *Rumba y Mambo* disco bar on top of the Danube. There, I developed a passion for Dominican bachata[4], which

[3] And for dealing with my attempts of speaking Chilean Spanish!

[4] Which, of course, I should call just *bachata*, but this is a topic for another day :)

helped me accept my body and enjoy even the darkest of the days. Thank you, Eva Vladimirova Manasieva and every member of Bachata con Sazon, for transmitting your passion to me: *esa es una vaina muy dura*. Your event in September 2024 was astonishing, and thanks to you, I got to know fabulous dancers such as El Guarachero, Ramon, Nushi, John, Junior, and JR el Artista. Thank you also, Karina and Cata, for teaching and helping me polish my style.

Latin transformation I will never forget my Melbourne flatmates Camila and Estefania, who introduced me to the Latin American culture. In particular, to the beauties of Colombia. They taught me the values of enjoying life, going out with friends, smiling and chilling. Thanks to you and the fantastic people I have met, like Connie, Leidy, Marcela, Sara, Sergio, Valentina, and Victoria. Even if I don't remember the taste of the *Vegan Ajaco*, I can say for sure that you changed my life. *Muchisimas gracias*.

During the summer of 2024, I had the chance to explore the Latin American community in Vienna. The amount of great time I had was indescribable; this was by far the most fun summer of my whole life. A big thanks go to *las chiquillas*, in particular to Marisol, Ignacia, Tamara, Melisa, and Daniela.

Chess Chess is one of those hobbies where you can meet everybody, and your opponent can be anyone, from an 8-year-old girl to an 80-year-old grandpa.

I experienced this when I arrived at the Melbourne Chess Club (MCC). I used to spend all my Saturdays and Tuesdays there (I remember that I even postponed a date because of a chess tournament; what a nerd...). At MCC, I met a group of people who made me feel special and appreciated. I hope to find another chess club as authentic and special as that one. Thanks for making my life easier, particularly for Alex Khamatgaleev[5], Colin, and Minnie (hopefully, we will be starting our postal correspondence chess match again). Special acknowledgement to Dr Michael Baron, who has been a great chess friend of mine!

A big shout-out to my chess buddy, Robin Coutelier. I am grateful for the many times I needed a distraction, and you were there. I'll never forget our chess tournament at the Vienna Central Cemetery (and the cool gadgets we got) and our frustration after the blitz tournaments that didn't go as expected. You made my life easier, *Merci beaucoup*.

Thanks also to Philipp Scheffknecht for organising the TU Wien Schach Club and the nice guys I met there. Thanks to Aris and Daniel. Btw, Paul & Alex, where the heck are you guys? >:D

Miscellaneous When I was 13 years old, I took the final exam of middle school. I got the lowest possible grade (*sufficiente*). After that, my maths teacher suggested that I attend an easier school, not a scientific one because math was not my strong suit

[5]The family name is too cool not to be mentioned.

(according to her). I still remember that day and my disappointment about that. Still, I decided to attend the scientific high school I wanted. After 18 years, I hope this final document proves her wrong. Maria Clara, I am glad that I did not believe you.

Geraldine Fitzpatrick changed my life, *sic et simpliciter*. I attended her course *From Surviving to Thriving* during the pandemic and the amount of things I learnt is unbelievable. I learnt about values, strenghts, gratitude, habits, boundaries, saying *no*. I started to know myself more. And thanks to your resources, I survived to the PhD. Thanks to Marta Cecchinato and its booklet *Digital Wellbeing* (extremely useful for whoever lives in 2025 with a smartphone). Thanks to Zoë J Ayres for her book *Managing Your Mental Health During Your PhD*, a precious resource for every PhD Student (and not only...).

Thanks to all my friends in Florence. In particular, I thank Alessandro and Guido for making my life so funny!

Thanks to all the others I forgot to mention!

Thank you to all VGSCO members, and a special thanks to Immanuel Bomze for solving the AMM problem 12480 with me. I will state it here since it is our "first paper" together:

American Mathematical Monthly problem

Problem 12480, *Proposed by F. Stanescu (Romania)*

Let f and g be two nonnegative functions on $[0,1]$ with $f(0) = g(0) = 1$, and let $h\colon [0,1] \to \mathbb{R}$ be nondecreasing. Suppose that f is convex, and g is concave. Prove

$$\int_0^1 g(x)\,\mathrm{d}x \int_0^1 f(x)h(x)\,\mathrm{d}x \geq \int_0^1 f(x)\,\mathrm{d}x \int_0^1 g(x)h(x)\,\mathrm{d}x$$

If you feel like having a solution, feel free to drop me an email :)

I acknowledge the funding provided by the doctoral program Vienna Graduate School on Computational Optimization through Austrian Science Foundation (FWF), under grant No.: W1260-N35 (https://vgsco.univie.ac.at/).

Last but not least, this journey would not have been possible without two people: my therapist, Carla, and my general practitioner, Ursula. Thank you for taking care of my mind and my body. Thank you also, Sara and Benedetta, for being my temporary therapists during the darkest period of my life.

Finally, I want to acknowledge that any journey has challenges; mine is no exception. I have made things very complicated, and I have hurt people. They were showing me something, and I couldn't see it. I beg your pardon and sincerely apologise for the issues I have created. I apologise to Anna F, Anna R, Clemens, Karen, Josephine, Ippolita, Raisa, Sasha, and Ziruo.

Reviewers If I were a serious person, I would probably say something dry and formal, along the lines of "Thanks to the two reviewers for their time...yada yada yada...".

However, as you might know, I don't like conventions (or maybe I'm just not a serious person).

Thank you, Andrea Schaerf. I remember our first meeting and told you the neighbourhood was working well. You interrupted me and asked, "Why is that so?". I was confused, and you kept asking "What makes the neighbourhood good? What is the reason?". This changed my point of view on the problems and the questions I should ask myself. I also remember you taught me how to properly explore the neighbourhood (not starting from the first index, to avoid bias). You made me a better scientist. Thank you.

Thank you, Maria Teresa Ortuño. You visited the University of Florence in 2018, introduced the topic of Optimisation, and proposed the double master's agreement. Taking that opportunity rocked me to my core. Little did I know that I would have ended up doing a PhD in Optimisation in your field. Thank you for your kindness and contagious smile.

Abstract

The Bus Driver Scheduling Problem (BDSP) is a combinatorial optimisation problem that involves assigning bus drivers to predetermined bus tours. The objective function takes into account the operational cost as well as driver satisfaction. This problem is heavily constrained due to stringent rules derived from laws and collective agreements.

In the literature, many methods exist to solve variants of the problem, but often they do not optimally solve larger instances. This causes the need for new, efficient methods to solve large-scale problem instances. In this thesis, we focus on a challenging variant of this problem based on the Austrian collective agreement. This variant has been introduced recently in the literature, and exact and heuristic methods have been developed. While these approaches give very good results, optimal solutions have not yet been found for large instances.

In this thesis, we proposed new metaheuristic approaches based on Tabu Search (TS) and Iterated Local Search (ILS). After thoroughly evaluating and comparing various algorithmic components, our experiments showed that TS scales very well to larger instances that are out of reach for exact methods.

Then, we presented a comprehensive study of a Large Neighbourhood Search (LNS) framework that uses Branch and Price (B&P) or Column Generation (CG) for the repair phase to solve the BDSP. We studied several variants of the LNS, as well as a deeper integration of B&P and LNS, in which we store the generated columns from the subproblems, and reuse them for other subproblems or to find better global solutions. Our analysis shows that this strategy improves several best-known solutions even more than B&P or LNS on their own, constituting a new state-of-the-art solution for this problem. This idea has still not been intensively explored and can be applied to other scenarios or to other optimisation problems.

Finally, we introduced an instance generator capable of producing new instances that are either synthetic or real-world-like. By doing so, we can expand the benchmark set of instances. Using those new instances in combination with the recently developed Instance Space Analysis (ISA) methodology, we gained insights into the strengths and weaknesses of our methods compared with the state-of-the-art algorithms.

Contents

CHAPTER 1

Introduction

The Bus Driver Scheduling Problem (BDSP) is a combinatorial optimisation problem with the goal of assigning bus drivers to vehicles in a company's regular daily operations. The scheduled bus tours are fixed in advance and will remain unchanged. The BDSP is one of the stages in the Transportation Network Planning System [IRDGM15]. The system consists of several stages, typically executed in sequence. The first stage, *Timetabling*, involves determining journeys start and end times at all bus stops. Next, *Vehicle Scheduling* focuses on assigning bus vehicles to specific bus tours. Following this, *Driver Scheduling* is responsible for defining daily shifts that ensure that all bus tours are covered. Finally, *Driver Rostering* assigns these shifts to bus drivers. These stages are depicted in Figure 1.1. The interested reader can find more information in the survey by Oscar Ibarra-Rojas et al. [IRDGM15] or in Avishai Ceder's book [Ced16].

The BDSP is a problem with clear practical relevance, and it has been studied since the early 1960's [Wre04, WR95]. The constraints of the BDSP depend on the country's legal regulation. In this thesis, we focus on a challenging variant of this problem based on the Austrian collective agreement for employees in private omnibus providers [Wir19]. In this variant, recently introduced by Lucas Kletzander and Nysret Musliu [KM20], the objective function considers the company's cost and the overall well-being of the employees. Here, the collective agreement has stringent rules requiring the drivers to take frequent breaks, with the option to split them into several parts. This translates to complex constraints rarely studied in the literature and prevents us from directly using algorithms developed for other variants of the BDSP. The practical relevance and the difficulty of the constraints make the problem challenging and interesting.

Regarding exact methods, the BDSP is often modelled as a Set Partitioning Problem, and Column Generation is used [SW88, LH16, PLP08, KMVH21]. However, due to the need to solve very large real-world problems in a reasonable time, several heuristic methods have been studied for the BDSP: some examples are Greedy [MT86], Tabu Search [SK01a, KMM22], Simulated Annealing [KM20], GRASP [DLFM11], CMSA

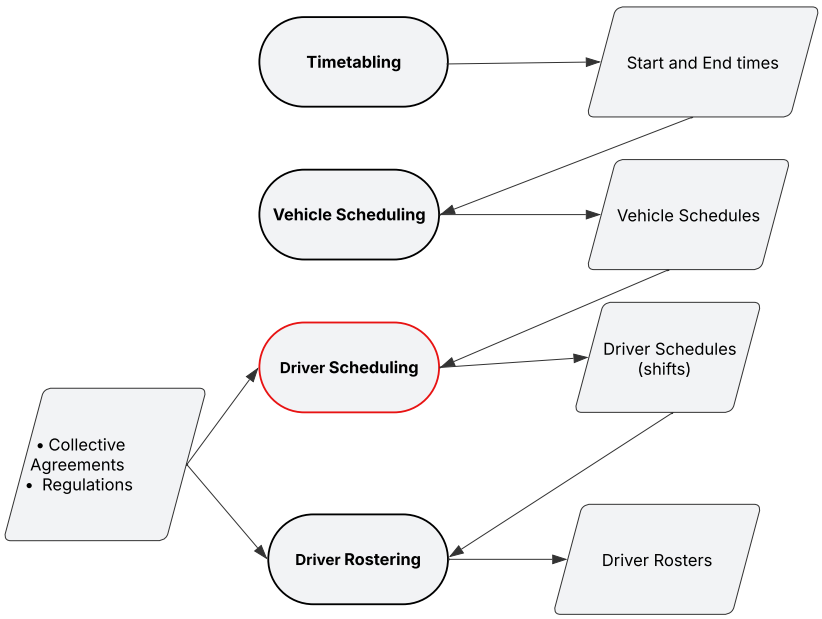


Figure 1.1: The Driver Scheduling Problem is one of the stages of the Transportation Planning System [IRDGM15]. Here, we show a simplified version.

[RKB+23], and Genetic Algorithm [LK03, LPP01]. Note that, in real life, a better solution would be beneficial for both employees and the company, since the objective function considers the well-being of the drivers. This also contributes to a more pleasant work environment that might lead to lower employee turnover and fewer sick days.

In this work, we propose new metaheuristic and hybrid algorithms for solving the BDSP, and we investigated their strengths and weaknesses. We evaluate our methods on a publicly available set of real-world-like instances and systematically analysed several moves, neighbourhoods, solvers, and variants. The experiments showed that our methods provide the best metaheuristics for many of the larger instances, improving several of the best-known results in the literature. Moreover, they can scale well to even larger instances recently introduced. Finally, with Instance Space Analysis, we gained insights into our methods' strengths and weaknesses and expanded our set of benchmark instances.

We now present the research questions that guide our investigation.

1.1 Research Goals

This section describes the research goals of my PhD programme and summarise the work done in the thesis.

1.1.1 Research Goal 1: Classical Metaheuristics for the BDSP

The first research question we address is:

RQ1

How do classical metaheuristics perform on the Austrian formulation of the BDSP? What are the most efficient components of these algorithms?

To answer this question, we first analysed and investigated different methods. For the BDSP, several metaheuristics and exact methods exist in the literature. Branch and Price [KMVH21] provides excellent results for instances with up to 60 bus tours. However, it fails to find good solutions for larger instances. In this case, Simulated Annealing and Hill Climbing provided the best-known solutions in the literature [KM20]. To improve them, we investigated methods based on Iterated Local Search and Tabu Search, which take into account several moves, parameters and perturbations. We studied and analysed the behaviour of heuristic methods based on these moves. We examined the impact of different solution components on the performance of the algorithms. We observed that while Simulated Annealing performs well with small instances, Tabu Search achieves the best results for larger instances, outperforming previous state-of-the-art methods.

1.1.2 Research Goal 2: Large Neighbourhood Search for the BDSP

The second research question we address is:

RQ2

How can we improve our methods by combining exact and heuristic approaches?

We combined the flexibility of heuristics with the completeness of exact methods for small subproblems. We use a hybrid method called Large Neighbourhood Search. This method iteratively modifies solutions using different destroy and repair operators. The main idea is to destroy part of a solution to obtain a subproblem that is easy to solve optimally or at least close to optimality. A chosen *destroy operator* selects and destroys a part of the solution. Then, a *repair operator* improves the incomplete solution. That is, it returns a feasible solution built from the destroyed one. The new solution is accepted if it is better than the previous one.

We also tested adaptivity, where the algorithm iteratively learns the best destroy operator to use and adapts the probability distribution accordingly.

Regarding the destroy operator, we consider three different operators that remove shifts, either uniformly, based on the cost of their shifts or associated with whole bus tours.

In the repairer phase, there are two good candidates from the literature: Branch and Price (B&P) and Column Generation (CG). Although Branch and Price (B&P) [KMVH21] returns the optimal solution for small instances, we observe that Column Generation

reaches very good solutions (with an optimality gap of around 1%) much faster. Therefore, we compare the performance of the sole Column Generation versus Branch and Price. This results in several Large Neighbourhood Search variants that we investigated.

We then propose a novel tight integration of Large Neighbourhood Search (LNS) and Column Generation (CG), where columns from each subproblem are aggregated and reused to improve the search significantly. Our approach constitutes the new state of the art, outperforming previous results across all sizes of instances. While this evaluation is specific to this version of the problem, the ideas used in B&P and LNS and their integration are generally applicable to complex personnel scheduling problems.

1.1.3 Research Goal 3: Instance Space Analysis for the BDSP

The third research question we address is:

RQ3

Which problem features best characterise the BDSP and its difficulty? How can we generate instances that are more diverse and representative?

Instance Space Analysis (ISA) [SMMn23] is a methodology designed to (a) support objective testing of algorithms and (b) assess the diversity of test instances.

We defined a set of instance *features* that capture the mathematical and statistical properties of the problem instance. For example, we investigated features related to the instance size, the distribution of the bus legs, or the flow between the bus stops.

Interestingly, features that appear in the literature on other problems can be translated into BDSP features. For example, we considered the proportion of bus legs whose length is larger than the average, like Kelvin Liu et al. [LSMC20] had done for the Bin Packing Problem.

We used the ISA to scrutinise the instance space of the BDSP to gain more insight into instances and algorithm performances. Given a set of instances and corresponding features, ISA highlights the regions of the instance space that are not covered by the available instances. To include these regions, we generated a number of new instances. To do so, we developed an instance generator that builds new instances using 44 parameters such as the number of bus stops and the average bus tour length. This allowed us to expand the previous set of instances from [KM20], which were limited and not diverse enough. we propose a set of features of the test instances to characterise their similarities and differences. We compared the two main state-of-the-art algorithms for the BDSP on real-world-like and random instances. We finally scrutinised the instance space and pointed out the regions that are not yet sufficiently covered. Although LNS was always performing better than Construct, Merge, Solve & Adapt (CMSA) on our initial set of instances, we could identify new types of instances where the opposite is true.

1.2 Summary of Contributions

We summarise the main contributions of this thesis:

- To efficiently solve real-world-like instances of the BDSP, we developed metaheuristic methods based on Tabu Search and Iterated Local Search. We analysed the impact of the parameters and components of those algorithms.
- We developed a novel hybrid method based on the Large Neighbourhood Search framework that combines the speed of heuristics with the quality of Branch and Price. The method achieves better results than the other methods in the literature.
- We propose a novel tight integration of Large Neighbourhood Search and Column Generation, where we store and reuse the columns from each subproblem to improve the search significantly.
- We developed an instance generator to generate new, larger, and diverse instances and make them publicly available.
- We propose a set of features of the test instances to characterise their similarities and differences.
- Using those features, we used Instance Space Analysis to gain insights into: (a) how algorithm performance is affected by instance properties, (b) the strengths and weaknesses of CMSA and LNS for different types of instances.

1.3 Publications

The results presented in this thesis have been included in the following publications:

- Tommaso Mannelli Mazzoli, Lucas Kletzander, Nysret Musliu, Kate Smith-Miles. **Instance Space Analysis for the Bus Driver Scheduling**, International Conference on the Practice and Theory of Automated Timetabling (PATAT), 2024 [MMKMSM24]
- Tommaso Mannelli Mazzoli, Lucas Kletzander, Nysret Musliu, Pascal Van Hentenrick. **Investigating Large Neighbourhood Search for Bus Driver Scheduling**, International Conference on Automated Planning and Scheduling (ICAPS) 2024 [MKHM24]
- Lucas Kletzander, Tommaso Mannelli Mazzoli, Nysret Musliu. **Metaheuristic Algorithms for the Bus Driver Scheduling Problem with Complex Break Constraints**, The Genetic and Evolutionary Computation Conference (GECCO) 2022 [KMM22]

Preprint

- Lucas Kletzander, Tommaso Mannelli Mazzoli, Nysret Musliu, Pascal Van Hentenrick. **Integrating Column Generation and Large Neighborhood Search for Bus Driver Scheduling with Complex Break Constraints**, https://arxiv.org/abs/2505.02485

1.4 Organisation

This thesis is organised as follows.

Section 2.3 provides the preliminary notions to understand the rest of the work.

Chapter 3 introduces the BDSP using the rules of the Austrian collective agreement for private omnibus providers and, in Chapter 4, we reviewed the literature on related problems.

Chapter 5 proposes metaheuristic methods based on Tabu Search and Iterated Local Search to approach the BDSP. We also evaluate the impact of the two methods on solution components and provide a new set of larger realistic benchmark instances that are publicly available. Chapter 6 introduces a hybrid method based on a Large Neighbourhood Search framework, where the repair operator is Column Generation. Here, we dropped the aim of optimally solving the subinstance with Branch and Price, and instead only used Column Generation on the root node to quickly get very good solutions to the subinstance. We expanded it even further, considering the reuse of the generated columns and adding a second thread that runs a global solver in the background to solve the master problem.

Chapter 7 uses Instance Space Analysis to show the weaknesses and strengths of the two solution methods. First, we extended an instance generator to generate varied real-world-like and random instances. This allows us to expand the previous set of instances from the literature. We then present a set of features that describe the hardness of the instances. The features represent the structure of the instance, such as the average break length for each vehicle or the distribution of bus tours in the city. Even if Large Neighbourhood Search outperforms Construct, Merge, Solve & Adapt in real-world-like instances, it does not perform as well for some random ones.

Finally, in Chapter 8, we gave concluding remarks and discussed future work.

CHAPTER 2

Preliminaries

About this chapter

We compress here the mathematical concepts and methodologies required to follow the thesis.

2.1 Notations, Symbols, and Definitions

Given a finite set A, let 2^A be the power set of A (the subsets of A): $2^A = \{X \mid X \subseteq A\}$. The cardinality (number of elements) of a set A is denoted by $|A|$.

We use the term *instance* consistently throughout this thesis. The following definition is adapted from *Combinatorial Optimization* by Christos H. Papadimitriou and Kenneth Steiglitz [PS98].

Definition 2.1 (Instance)**.** *An **instance of an optimisation problem** (or, briefly, instance) is a pair $(\mathcal{X}, z)$ where $\mathcal{X}$ is a finite set (named "the solution space") and $z\colon \mathcal{X} \to \mathbb{R}$ is the objective function that we aim to minimise.*

The problem is to find the global optimum.

Definition 2.2 (Global optimum)**.** *Given an instance $(\mathcal{X}, z)$ of an optimisation problem, a solution $S^* \in \mathcal{X}$ is called a **global optimum** if for all $S \in \mathcal{X}$ it holds that $z(S^*) \leq z(solution)$.*

Remark

Often $\mathcal{X}$ and z are given "implicitly" in terms of two algorithms $A_{\mathcal{X}}$ and A_z. The algorithm $A_{\mathcal{X}}$ takes as input an object S, a set P of parameters and decides whether S is an element of $\mathcal{X}$. The algorithm A_z, instead, given a solution[a] $S \in \mathcal{X}$ and another set of parameters Q, returns the value of $z(S)$. In most cases (including ours), we assume that $A_{\mathcal{X}}$ and A_z are polynomial-time algorithms, but there are exceptions [PS98, Problem 17(a) at pag 379].

[a] As it is very usual in the field of Combinatorial Optimisation, we use the word *solutions* to refer to *feasible points* (even though they are **not** global optima)

The main idea of the neighbourhood of a solution S is that it is a set of solutions *close* to S. Needless to say, the definition of closeness depends on the problem.

Definition 2.3 (Neighbourhood)**.** *Given an instance $(\mathcal{X}, z)$ of an optimisation problem, a* ***neighbourhood*** *is a function $N\colon \mathcal{X} \to 2^{\mathcal{X}}$ that maps a solution into a set of solutions. If $S \in \mathcal{X}$, $N(S)$ is called a* ***neighbourhood*** *of S.*

Example of a neighbourhood

Let $\mathcal{X} = \{0,1\}^n$ be the set of binary vectors of length n. Let $\mathit{flip}\colon \mathbb{N} \times \mathcal{X} \to \mathcal{X}$ be the function $\mathit{flip}(i, S)$ that flips the i-th component of the solution S:

$$\mathit{flip}(i, S) = (S_1, S_2, \ldots, S_{i-1}, \bar{S}_i, S_{i+1}, \ldots S_n).$$

flipped element in position i

This induces a neighbourhood $N_{\mathit{flip}}(s) = \{\mathit{flip}(i, s) \mid i \in \{1, 2, \ldots, n\}\}$. For example, for $n = 3$ and $s = (0, 1, 0)$, we have

$$N_{\mathit{flip}}\Big((0,1,0)\Big) = \Big\{(\,1\,, 1, 0), (0, \,0\,, 0), (0, 1, \,1\,)\Big\}.$$

The concept of a neighbourhood naturally leads to the definition of local optimum.

Definition 2.4 (Local Optimum)**.** *Given an instance $(\mathcal{X}, z)$ of an optimisation problem and a neighbourhood N, a feasible solution $\hat{S} \in \mathcal{X}$ is a* ***local optimum with respect to*** *N if $z(\hat{S}) \leq z(S)$ for all $S \in N(\hat{S})$.*

It is important to note that, in general, there is no guarantee of the quality of the local optimum obtained.

2.2 Instance Space Analysis

Instance Space Analysis (ISA) is a methodology proposed by Kate Smith-Miles et al. in 2014 [SMBWL14] that extends the algorithm selection problem framework of John R. Rice. [Ric76]. The current form is described in the article from 2023 by Kate Smith-Miles and Mario Andrés Muñoz [SMMn23].

In Instance Space Analysis (ISA), instances are represented as vectors of features. The instances are then projected onto the 2D plane to separate hard and easy instances. Figure 2.1 illustrates the Instance Space Analysis framework.

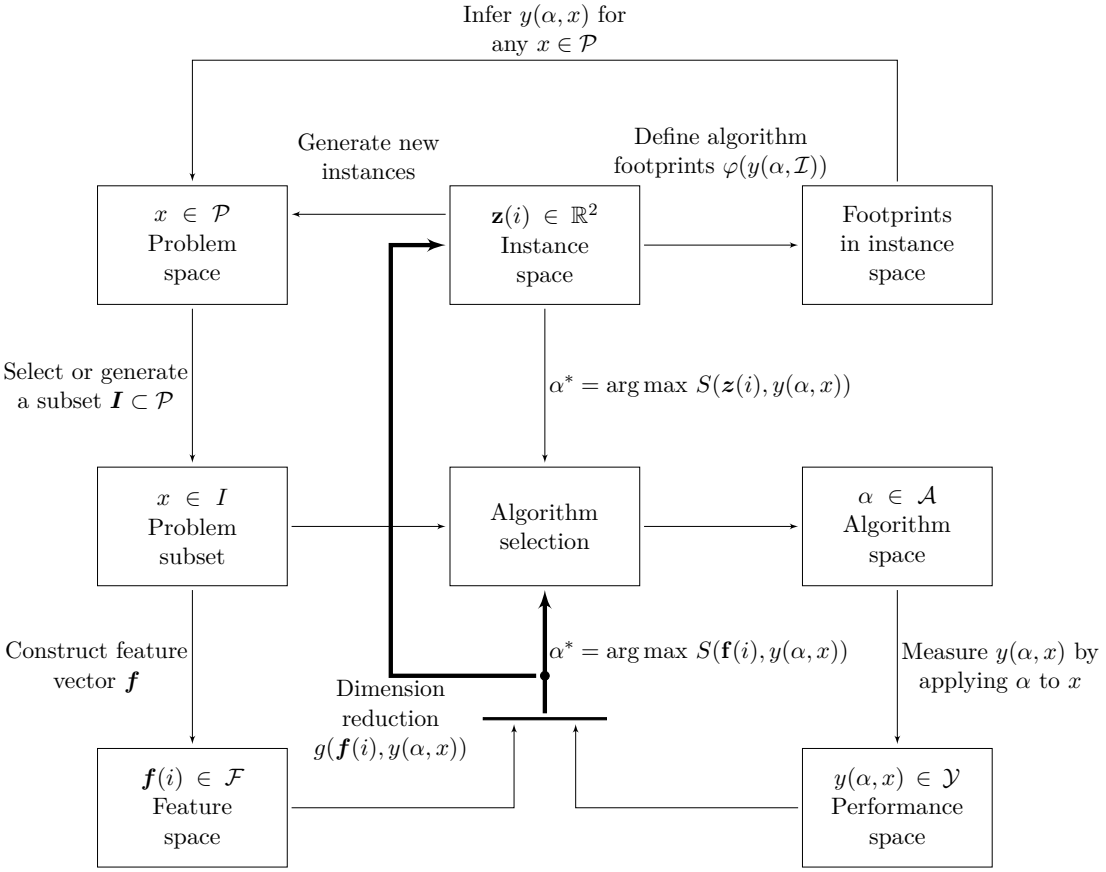


Figure 2.1: ISA framework [SMMn23]

The *problem space* $\mathcal{P}$ contains all the theoretically possible instances of an optimisation problem. However, in practice we only consider a (smaller) subset of instances $\mathbf{I} \subset \mathcal{P}$, for which we have experimental results.

The first component of the meta-data are some chosen *features*, used to characterise the mathematical and statistical properties of the instances that (1) describe the similarities and differences between instances in (2) have correlation with the performance of one or

more algorithms. For a given instance $x \in \mathbf{I}$, we calculate the feature vector $\boldsymbol{f}(i)$, which represents an instance in the *feature space*, $\mathcal{F}$.

The second component is the *performance measure*. For each algorithm $\alpha \in \mathcal{A}$ (the algorithm space) and each instance $x \in \mathbf{I}$, the performance measure $y(\alpha, x)$ measures the "goodness" of the algorithm on that specific instance. This could be, for example, the objective function value obtained for a fixed computational budget. The term $y(\alpha, x)$ is an element of a vector that describes the *performance space*, $\mathcal{Y}$.

Together, the features and performance measures for all the instances in I, and all algorithms in $\mathcal{A}$ constitute the *meta-data*. We represent this meta-data as two matrices $\boldsymbol{F} = [\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n] \in \mathbb{R}^{m \times n}$ and $\boldsymbol{Y} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n] \in \mathbb{R}^{a \times n}$, where m is the number of features, n is the number of problem instances, and a is the number of algorithms. Hence, the meta-data is the set $\{\boldsymbol{F}, \boldsymbol{Y}\}$.

In the original framework proposed by Rice in 1976 [Ric76], a selection mapping S was learned directly from features and performance. Later, in the expanded framework introduced by Smith-Miles et al. in 2014 [SMBWL14], and extended by Smith-Miles and Muñoz in 2023 [SMMn23], instances are projected from the feature space into a lower-dimensional 2D space using the dimension reduction $g\left(\boldsymbol{f}(x), y(\alpha, x)\right)$. It aims to yield an optimal projection that looks for linear trends in both features and algorithmic performance across the resultant instance space, to gain interpretable insights. This allows us to get a visualisation and enables a more detailed evaluation of algorithmic performance, as well as algorithm selection based on the position of an instance in the instance space.

In the conceptual framework delineated in the preceding section, the ISA methodology involves six fundamental steps:

1. Acquiring experimental meta-data for a designated set of instances $\mathbf{I}$ ran across a set of algorithms $\mathcal{A}$. This includes capturing feature values $\boldsymbol{F}$ for all instances and recording performance metrics $\boldsymbol{Y}$ for all algorithms across all instances.

2. Creating an instance space through a feature selection process applied to the meta-data $\{\boldsymbol{F}, \boldsymbol{Y}\}$, with consideration for a user-defined benchmark for optimal performance, encompassing its theoretical boundaries.

3. Employing machine learning techniques to generate predictions for automated algorithm selection.

4. Establishing algorithm footprints and evaluating associated metrics.

5. Analysing the instance space to extract insights and assess the adequacy of the available meta-data.

6. Generating supplementary meta-data as needed, and iterating from Step 2 onwards, or concluding the process if no further iterations are warranted.

2.3 Experiments

Note on reporting experiments Regarding the experiments, we used the same methodology discussed in the book of Mike Preuss *Experimentation in Evolutionary Computation* [Pre15]. He proposes to organise the presentation of experiments into seven parts. I report it here for completeness.

1. *Research question*
   Briefly names the matter dealt with, the (possibly very general) objective, preferably in one sentence. This is used as the report's "headline" and related to the primary model.

2. *Pre-experimental planning*
   Summarizes the first—possibly explorative—program runs, leading to *Task* (3) and *Setup* (4). Decisions on employed benchmark problems or performance measures should be taken according to the data collected in preliminary runs. The report on pre-experimental planning should also include negative results, e.g., modifications to an algorithm that did not work or a test problem that turned out to be too hard, if they provide new insight.

3. *Task*
   Concretizes the question in focus and states scientific claims and derived statistical hypotheses to test. Note that one scientific claim may require several, sometimes hundreds, of statistical hypotheses. In case of a purely explorative study, as with the first test of a new algorithm, statistical tests may not be applicable. Still, the task should be formulated as precisely as possible. This step is related to the experimental model.

4. *Setup*
   Specifies problem design and algorithm design, including the investigated algorithm, the controllable and the fixed parameters, and the chosen performance measuring. It also includes information about the computational environment (hard- and software specification, e.g., the packages or libraries used). The information provided in this part should be sufficient to replicate an experiment.

5. *Results/Visualization*
   Gives raw or produced (filtered) data on the experimental outcome and additionally provides basic visualizations where meaningful. This is related to the data model.

6. *Observations*
   Describes exceptions from the expected, or unusual patterns noticed, without subjective assessment or explanation. As an example, it may be worthwhile to look at parameter interactions. Additional visualizations may help to clarify what happens.

7. *Discussion*
   Decides about the hypotheses specified in 3, and provides necessarily subjective interpretations of the recorded observations. Also places the results in a wider context. The leading question here is: What did we learn?

It is important to divide parts 6 and 7, to facilitate different conclusions drawn by others, based on the same results/observations. Note that all of these parts are already included in current good experimental reports. However, they are usually not separated but wildly mixed.

CHAPTER 3

The Bus Driver Scheduling Problem

About this chapter

This chapter introduces the Bus Driver Scheduling Problem with Complex Break Constraints. This is a challenging scheduling problem originating from the general Transportation Planning System.

3.1 Background

We can distinguish three goals of the BDSP. The *first* goal is to define daily shifts that cover predetermined bus tours and meet the transportation system's demand.

Fatigue is a major safety concern for drivers, and they must have enough time to rest and recharge to avoid accidents and maintain a high level of safety for passengers. This is why the *second* goal of the BDSP is to respect all the requirements from the collective agreement and legal regulations. For example, we must ensure that the drivers have sufficient driving breaks during their shifts.

The *third* goal of the BDSP is to reduce cost and maximise effectiveness. This means not only saving money for the bus company, but also minimising the dissatisfaction of the drivers. For example, breaks that last more than three hours are usually poorly regarded by the drivers (they are unpaid), and we consider this in the objective function.

This study considers regulations from two documents: The first one is the *European Regulation (EC) No 561/2006* [Com25] that describes rules on driving times, breaks and rest periods for vehicle drivers of public transport. The second document is the *Austrian Collective Agreement for employees in private omnibus providers* [Wir19] using the rules

for regional lines up to 50 km. This is particularly strict regarding the break regulations, which makes the BDSP extremely constrained and challenging to solve.

Although we could rename this variant the Bus Driver Scheduling Problem with Austrian Regulations (or Bus Driver Scheduling Problem with Complex Constraints), for clarity we simply refer to it as the *Bus Driver Scheduling Problem*.

3.2 Problem Description

This section provides a formal specification of the BDSP. This section is partially taken from previous work in the literature [KM20, Kle22].

3.2.1 Problem Input

The input of the problem consists of three pieces of data:

- **Positions and Distance Matrix**: A finite set $P \subseteq \mathbb{N}$ of *positions* or bus stops. A time distance matrix $D = (d_{ij}) \in \mathbb{N}^{|P|\times|P|}$ where d_{ij} represents the time needed for a driver to go from position i to j when not actively driving a bus. If no transfer is possible, then we set $d_{ij} = +\infty$[1]. When $i \neq j$, d_{ij} is referred to as *passive ride time*. In contrast, d_{ii} represents the time required to switch tour at the same position, but is not considered passive ride time.
- **Start and End Work**: for every position $p \in P$, two integer values $startWork_p$ and $endWork_p$ respectively represent the time required to start or end a shift at that position.
- **Bus Legs**: A set L of Bus Legs, where each leg $\ell \in L$ is defined as a 5-tuple:

  $$\ell = (tour_\ell, startPos_\ell, endPos_\ell, start_\ell, end_\ell),$$

  representing the trip of a vehicle between two stops at a certain time:

  - $tour_\ell \in \mathbb{N}$ is the ID of the vehicle.
  - $startPos_\ell, endPos_\ell \in P$ the starting and the ending positions of the leg.
  - $start_\ell \in \mathbb{N}$ is the time at which the vehicle departs from position $startPos_\ell$.
  - $end_\ell \in \mathbb{N}$ is the time at which the vehicle arrives to position $endPos_\ell$.

  A bus leg ℓ is uniquely defined by its pair of start time $start_\ell$ and bus tour $tour_\ell$.

  The driving time of a bus leg is $length_\ell = end_\ell - start_\ell$. Bus Legs belonging to the same Bus Tour t do not overlap, which means that the intervals $(start_\ell, end_\ell)$ for ℓ with $tour_\ell = t$ are disjoint. Note that all times are expressed in minutes, and time the horizon spans 24 hours.

[1] I apologise to any mathematician reading this. In fact, in such cases, we set $d_{ij} = 999\,999 = 10^6 - 1$.

Table 3.1 shows a small example of a set of six legs. The first vehicle starts at time 360 (6:00 am) at position 0 and travels between positions 1 and 2 with waiting times in between. Finally, it returns to position 0.

Table 3.1: A toy example with two bus tours from [KMM22]

$tour_\ell$	$startPos_\ell$	$endPos_\ell$	$start_\ell$	end_ℓ
1	0	1	360	395
1	1	2	410	455
1	2	1	460	502
1	1	0	508	540
2	0	3	360	400
2	3	0	415	500

We can think of the set of bus legs L as a totally ordered set (Lemma 1).

Lemma 1. *Let $\preceq$ be the relation on L defined as follows: for any $\ell_1, \ell_2 \in L$, write $\ell_1 \preceq \ell_2$ if either*

- $start_{\ell_1} < start_{\ell_2}$, *or*
- $start_{\ell_1} = start_{\ell_2}$ *and* $tour_{\ell_1} \leq tour_{\ell_2}$

Then $\preceq$ is an order relation, and $(L, \preceq)$ is a totally ordered set.

Proof.

- *Reflexivity*: For each $\ell \in L$ we have $start_\ell = start_\ell$ and $tour_\ell = tour_\ell$. So $\ell \preceq \ell$.
- *Antisymmetry* Let $\ell_1, \ell_2 \in L$ be such that $\ell_1 \preceq \ell_2$ and $\ell_2 \preceq \ell_1$. This implies that $start_{\ell_1} = start_{\ell_2}$, $tour_{\ell_1} \leq tour_{\ell_2}$, and $tour_{\ell_2} \leq tour_{\ell_1}$. So $\ell_1 = \ell_2$.
- *Transitivity* Let $\ell_1, \ell_2, \ell_3 \in L$ be such that $\ell_1 \preceq \ell_2$, and $\ell_2 \preceq \ell_3$. Then there are four cases:
  1. $start_{\ell_1} < start_{\ell_2} < start_{\ell_3}$. Then $start_{\ell_1} < start_{\ell_3}$ and therefore $\ell_1 \preceq \ell_3$.
  2. $start_{\ell_1} = start_{\ell_2} < start_{\ell_3}$ so $\ell_1 \preceq \ell_3$.
  3. $start_{\ell_1} < start_{\ell_2} = start_{\ell_3}$ so $\ell_1 \preceq \ell_3$.
  4. $start_{\ell_1} = start_{\ell_2} = start_{\ell_3}$ and $tour_{\ell_1} \leq tour_{\ell_2} \leq tour_{\ell_3}$. This implies $\ell_1 \preceq \ell_3$.
- *Totally ordered*: For any $\ell_1, \ell_2 \in L$, exactly one of the following holds:
  - $start_{\ell_1} < start_{\ell_2}$,

- $start_{\ell_1} > start_{\ell_2}$, or
- $start_{\ell_1} = start_{\ell_2}$.

In the first two cases, $\ell_1 \preceq \ell_2$ or $\ell_2 \preceq \ell_1$ follows directly. If $start_{\ell_1} = start_{\ell_2}$, then $tour_{\ell_1} \leq tour_{\ell_2}$ or $tour_{\ell_1} \geq tour_{\ell_2}$. Thus, $\ell_1 \preceq \ell_2$ or $\ell_2 \preceq \ell_1$. Hence, $(L, \preceq)$ is totally ordered. □

3.2.2 Solution

A *solution* S to the BDSP is an assignment of drivers to bus legs. Formally, it is represented as a *set partition* of L, denoted as $S = \{s_1, s_2, \ldots, s_n\}$, where each block $s_i \in S$ is denoted as a *shift*. Each shift s_i represents a subset of bus legs that are assigned to a single driver. A priori, the number of shifts n is not given. Nevertheless, we can imagine to set it as large as we need in order to get a feasible solution. The largest possible partition occurs when each shift contains only one leg, implying that $|S| \leq |L|$. A lower bound for $|S|$ will be discussed in Lemma 2.

Equivalently, it can be useful to think about *shifts* as *the work scheduled to be performed by a driver in one day* [Wre04]. For convenience, we may sometimes refer to the shift s as the *driver* s.

Note that, since the set L is totally ordered (Lemma 1), the notion of *consecutive* legs in a shift s is well defined. Moreover, a solution is also totally ordered by the order induced by the bus legs.

A solution is feasible if it satisfies the following criteria:

- For every shift s in the solution, if $i, j \in s$ are consecutive bus legs and $tour_i \neq tour_j$ or $endPos_i \neq startPos_j$, then the driver must have enough time to cover the distance between $endPos_i$, and $startPos_j$. This rule is expressed by the constraint

$$start_j \geq end_i + d_{endPos_i, startPos_j}.$$

  This implicitly guarantees no overlapping bus legs within a single shift, since $d_{p,q} > 0$ for every $p, q \in P$.

- Each shift must satisfy all hard constraints depending on the laws specified in the next sections.

3.2.3 Constraints

This section describes the constraints of our BDSP variant, derived from the Austrian collective agreement.

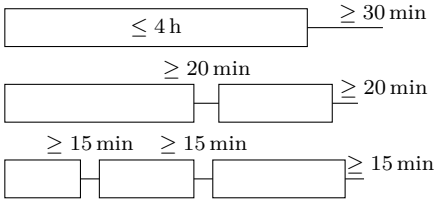


Figure 3.1: Driving time constraints and required break options [KM20].

Driving time

$$D_s = \sum_{i \in s} length_i = \sum_{i \in e} (end_i - start_i) \qquad \forall s \in S \qquad (3.1)$$

$$D_s \leq D_{\max} = 540 \qquad \forall s \in S \qquad (3.2)$$

Equation (3.1) defines the *driving time* D_s of a shift e. Constraints Equation (3.2) set the upper bound of D_s of nine hours. The driving time is subject to additional rules regarding driving breaks. The length of a *driving break* between two consecutive bus legs i and j is

$$diff_{ij} = start_j - end_i.$$

The driving break can be split in multiple parts, all of which must be completed *before* the cumulative driving time without a break reaches 4 h:

- One driving break of at least 30 min.
- Two driving breaks of at least 20 min each.
- Three driving breaks of at least 15 min each.

These driving rules are shown in Figure 3.1. Once we reach all required breaks, a new block of at most 4 h begins.

Total Time

$$T_s = end_\ell + endWork_\ell - (start_f - startWork_f) \qquad \forall s \in S \qquad (3.3)$$

$$T_s \leq T_{\max} = 840 \qquad \forall s \in S \qquad (3.4)$$

Equation (3.3) defined the *total time* of a shift s: it is the span from the start of work until the end of work, where f is the first leg and ℓ is the last leg in the shift s. Equation (3.4) sets the upper bound of T_s: no driver can work more than fourteen hours.

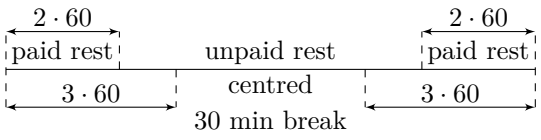


Figure 3.2: Rest break positioning [KMM22]

Shift Splits.

An important concept is the one of **shift splits**. We say that the shift e contains a shift split if there exists a pair of consecutive legs $i, j \in e$ such that the break between them satisfies

$$start_j - end_i - r_{endPos_i, startPos_j} \geq 180.$$

Here, $r_{p,q} = d_{p,q}$ denotes the time required for a passive ride between position p and q, where $r_{p,q} = d_{p,q}$ if $p \neq q$ and $r_{p,p} = 0$. Hence, shift splits refer to breaks longer than three hours. As these breaks are unpaid, they are generally poorly regarded by bus drivers. This plays a role in designing the objective function.

Denote by $split_s$ the number of shift splits in shift s and by $splitTime_s$ the total duration of these shift splits. A shift split resets the driving time (i.e., it counts as a *driving* break). A shift contains up to two shift splits.

Rest Break

Let i, j be two consecutive bus legs in a shift. The break between these two legs can be represented as the time interval $[end_i, start_j]$, and its length as $diff_{ij} = start_j - end_i$. We denote with $diff'_{ij}$ the length of a *rest break*:

$$diff'_{ij} = \begin{cases} diff_{ij} - r_{endPos_i, startPos_j} & \text{if } 15 \leq diff_{ij} - r_{endPos_i, startPos_j} \leq 180 \\ 0 & \text{otherwise} \end{cases}$$

If the duration of a (partial) rest break is at least at least 15 min long, the break is considered unpaid if it does not fall within the first 2 or last 2 hours of the shift. The maximum amount of unpaid rest breaks ($upmax_s$) is limited, as shown in Figure 3.2:

- If 30 consecutive minutes of rest break do not intersect the first 3 h or the last 3 h of the shift, at most 1.5 h of unpaid rest are allowed and therefore we set $upmax_s = 75$.
- Otherwise, at most 1 h of unpaid rest is allowed, and therefore we set $upmax_s = 60$.

In formulas,

$$upmax_s = \begin{cases} 90 & \text{if a 30-minute consecutive rest break does not intersect} \\ & \text{the first 3 h or the last 3 h of the shift,} \\ 60 & \text{if there is a 30-minute break centered between the first} \\ & \text{3 h and the last 3 h of the shift,} \\ 0 & \text{otherwise.} \end{cases}$$

Rest breaks beyond this limit are paid. We denote by $unpaid_s$ the sum of the length of unpaid rest breaks.

Working time

$$W_s = T_s - splitTime_s - \min\{unpaid_s, upmax_s\} \qquad \forall s \in S \qquad (3.5)$$

$$W_s \leq W_{\max} = 600 \qquad \forall s \in S \qquad (3.6)$$

Equation (3.5) defines the *working time* of a shift s. Equation (3.6) sets the upper bound of W_s as ten hours.

The working time is subject to additional rules regarding rest breaks. The minimum rest break depends on the working time:

- $W_s < 300$: no rest break required.
- $300 \leq W_s \leq 540$: at least one 30-minute break.
- $W_s > 540$: at least one 45-minute break.

Rest breaks may be split into smaller parts. If a break is split, one part must be at least 30 min in duration, and any additional part must be at least 15 min. The first part of the break must occur within the first six hours of working time.

3.2.4 Objective function

Let S be a solution. The objective function, that we want to minimise, is defined as follows [KM20, KMVH21]:

$$z(S) = \sum_{s \in S} \left(2W'_s + T_s + ride_s + 30change_s + 180split_s\right), \tag{3.7}$$

where, for every shift s:

- $W'_s = \max\{W_s, 390\}$, where W_s is the working time defined by Equation (3.5). This objective ensures that drivers are paid for at least 6.5 hours (390 minutes) of work, even if they work less.
- T_s is the total time of the shift as defined in Equation (3.3).
- $ride_s$ is the sum of passive ride times between consecutive legs.
- $change_s$ is the number of *tour changes* that is, the number of occurrences of consecutive bus legs $i, j \in s$ with $tour_i \neq tour_j$.
- $split_s$ is the number of shift splits.

The coefficients of the linear combination were determined by a previous work [KM20] based on preferences agreed upon by different stakeholders at Austrian bus companies and employee scheduling experts.

As argued by Kletzander and Musliu [KM20], practical schedules must not only consider operating costs, but also the well-being of the drivers. For this reason, *change*, *split* and total time T_s are included in the objective function.

In Figure 3.3 we picture an example of a shift with three bus legs.

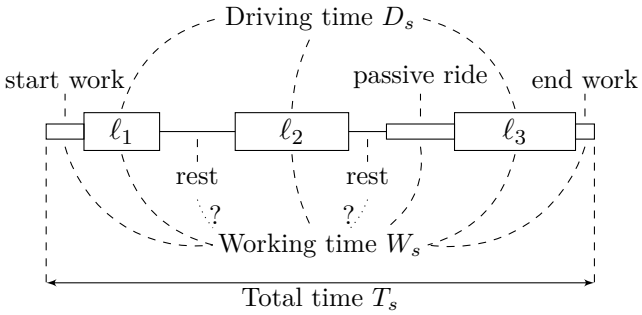


Figure 3.3: Example shift $s = \{\ell_1, \ell_2, \ell_3\}$ [KMM22]

Upper bound of the objective function

Consider an instance of the BDSP. A *trivial* solution consists of assigning one shift to each leg. Let $L = \{\ell_1, \ell_2, \dots, \ell_n\}$ be the set of bus legs and let S be the solution defined as $S = \{\{\ell_1\}, \{\ell_2\}, \dots, \{\ell_n\}\}$.

The solution S is feasible and the objective function eq. (3.7) is

$$\begin{aligned} z(S) &= \sum_{s \in S} (T_s + 390) \\ &= 780 \cdot |L| + \sum_{\ell \in L} (startWork(startPos(\ell)) + endWork(endPos(\ell)) + length_\ell)\,. \end{aligned}$$

This gives us an **upper bound** for the optimal solution S^*:

$$z(S^*) \leq 780 \cdot |L| + \sum_{\ell \in L} \Big(startWork\big(startPos(\ell)\big) + endWork\big(endPos(\ell)\big) + length_\ell \Big).$$

We are now ready to evaluate a lower bound for the number of shifts of a solution $|S|$. This proof is adapted and expanded from previous work [FMT87].

Lemma 2. *Consider one instance of the BDSP and define*

$$\lambda_1 := \left\lceil \frac{1}{D_{\max}} \sum_{\ell \in L} length_\ell \right\rceil \qquad \text{and} \qquad \lambda_2 := \max\Big\{ a(t_1) + a(t_2) : |t_1 - t_2| > T_{\max} \Big\},$$

where $a(t) = |\{\ell \in L : start_\ell \leq t \leq end_\ell\}|$ is the number of bus legs at time t that are active, i.e., their correspondent bus tour (vehicle) is currently driving.

Then, for every feasible solution S, it holds that $\max\{\lambda_1, \lambda_2\} \leq |S|$.

Proof. let S be a feasible solution. Summing the length of every bus leg in L, we have

$$\sum_{\ell \in L} length_\ell = \sum_{s \in S} D_s \leq |S| \cdot D_{\max}. \tag{3.8}$$

where D_s and $D_{\max}$ have been defined in Equation (3.1) and Equation (3.2). This implies that

$$|S| \geq \left\lceil \frac{1}{D_{\max}} \sum_{\ell \in L} length_\ell \right\rceil =: \lambda_1.$$

A second lower bound takes into account the number of active bus legs $a(t)$ at time t. Clearly, $|S| \geq a(t)$ for any given time t, because each active leg must have a driver assigned to it. However, this lower bound can be improved using the fact that for every time t, no active employee at time t can be active at time $t + T_{\max}$ or $t - T_{\max}$, because of Equation (3.4). Therefore,

$$|S| \geq \max\Big\{ a(t_1) + a(t_2) : |t_1 - t_2| > T_{\max} \Big\} =: \lambda_2.$$

Hence, $|S| \geq \max(\lambda_1, \lambda_2)$. □

Note that two optimal solutions can have different numbers of shifts, as shown by the following example.

Two optimal solutions with different number of shifts

Consider the instance with

- one position p, and $d_{pp} = 10$.
- $startWork_p = endWork_p = 0$.
- two bus legs $L = \{\ell_1, \ell_2\}$:

$$\ell_1 = (1, 100, 200, p, p),$$
$$\ell_2 = (1, 800, 900, p, p).$$

Consider the two solutions $S_1 = \{\ell_1, \ell_2\}$ and $S_2 = \{\{\ell_1\}, \{\ell_2\}\}$. Using Equation (3.7), we can compute

$$z(S_1) = 800 + 2 \cdot 390 + 180 = 1760$$
$$z(S_2) = 2 \cdot 390 + |\ell_1| + 2 \cdot 390 + |\ell_2| = 1560 + 200 = 1760$$

The two solutions S_1 and S_2 have the **same** objective function value, but **distinct** number of shifts.

3.3 Benchmark Instances

The original instances cannot be publicly shared, due to agreements with the companies. That is why an instance generator was developed, able to generate real-world like instances that follow a similar distribution as the original ones. For our work, we use a set of 50 real-world like instances [KM20]. There 50 instances are divided in 10 sizes, ranging from around 10 tours to around 100 bus tours. The instance name is `realistic_xx_y` where `xx` is the size of the instance and `y` is the instance number. All instances are publicly available: https://cdlab-artis.dbai.tuwien.ac.at/papers/sa-bds/.

In addition to the 50 instances in the literature, we generated 15 new test instances based on real-world-like distributions that range in size from 148 tours (more than 1300 bus legs) to 250 tours (about 2300 bus legs). Instances of this size occur in practice when larger (Austrian) cities are considered in their whole, which presents more potential savings compared to only dealing with subsections of the city.

CHAPTER 4

Related Work

Since this problem strongly depends on local regulations, many variants in the literature exist from different cities worldwide. We briefly describe some of these variants here to illustrate the diverse rules, regulations, and constraints one might encounter.

4.1 Variants

Ravenna (1986) In 1986, Silvano Martello and Paolo Toth [MT86] published a paper about the Bus Driver Scheduling Problem in Ravenna, Italy. Their version aims to minimise the number of shifts. Here are some of the constraints from their paper:

- Maximum total time $T_s \leq T_{\max}$ for every shift s,
- Maximum driving time $D_s \leq D_{\max}$ for every shift s,
- The average value of driving times cannot exceed $\bar{D}_{\max}$,
- If a shift s has total time T_s greater than a value $\bar{T}$, then it is considered *long*. At most λ long shifts are allowed.
- Shifts starting by a time m' (morning shifts) must finish by a time m''.
- *Meal breaks* must be given to drivers at noon and in the evening. The length of a meal break must be at least $\bar{m}$. Each driver starting his shift by time n' and finishing it after a time n'' must have a meal break in the period $[n' - \bar{m}, n'' + \bar{m}]$.

As for the complexity, in 1987, Silvano Martello, Paolo Toth and Matteo Fischietti showed [FMT87] that the recognition version[1] of their problem is **NP**-hard. Their

[1]The *recognition version* of an optimisation problem is a decision (binary) problem that we can state as: Given an instance (S, f) and an integer $L > 0$, is there a feasible solution $x^* \in S$ such that $f(x^*) \leq L$?

version only considers Total Time constraints, meaning each shift cannot be longer than a value $T_{\max}$. They proved that the Vertex Colouring Problem reduces to an instance of the BDSP.

Two years later, the same authors showed [FMT89] that the recognition version of their BDSP with only driving time constraints is also an **NP**-hard problem, as the Bin Packing problem reduces to the BDSP.

Note that we cannot directly use these complexity results since their problem differs from ours (for instance, we are not interested in minimising the number of shifts).

Leeds (1988) Research about the BDSP in Leeds (UK) has been led since the 1960s by Anthony Wren, one of the major contributors to the BDSP. In 1988, Barbara Smith, both independently [Smi88] and in collaboration with Anthony Wren [SW88], published a paper where they used a Set Covering Problem formulation in a system called IMPACS. The system and its successor, TRACS II [FPW02, WFK+03] which is known to typically achieve savings of 1–5% of driver costs[FPW02], were supplied in British cities such as London, Manchester, Cleveland, Strathclyde, and Portsmouth.

Montreal (1990) HASTUS is a family of products that aim to solve several problems from the Public Transport that has its roots in Montreal (Quebec, Canada). The main idea is described in a paper by Blais et al.[BLR90]. The HASTUS system has been successfully deployed in nearly 40 major cities worldwide.

Tenerife (2023) The work of Guillermo Esquivel-González et al. in 2023 [EGSNL23] analyses a version of the problem that arose in Tenerife, Spain. Some of the characteristics of their problem seem fairly similar to ours: the company's collective agreement requires that all drivers take a 25-minute break on each shift between the third and fifth hour of their shift. However, their objective function is entirely different and does not aim to minimise operational costs. Instead, the goal is to maximise the number of passengers picked up.

Singapore (2024) In Singapore, there are autonomous buses operated without human drivers. However, safety drivers still supervise the operations and help in case of unexpected events [WCM24]. Using this setting, Wang et al. studied [WCM24] the *Robust Safety Driver Scheduling*: a model that considers uncertainties such as primary delays caused by traffic jams. The researchers modified the Set Partitioning Problem, introducing a term for maximum total delay within uncertainties. To solve this problem, they developed a Branch-and-price-and-cut algorithm that reached better results than CPLEX-based Branch and Price.

4.2 Exact methods

Exact algorithms are usually the initial approach for solving an optimisation problem. As stated by Anthony Wren [Wre04], the BDSP research dates back to the 1960s, but practical applications of exact methods remained limited until the late 1970's [WM14, Sec 4].

For this problem, the main idea of these methods is to formulate the BDSP as either a Set Partitioning Problem or a Set Covering Problem. Let $\mathcal{P} \subseteq 2^L$ be the collection of all possible feasible shifts; its size $|\mathcal{P}|$ is a large but finite number. A feasible solution S is a subset $S \subseteq \mathcal{P}$ such that every leg $\ell \in L$ belongs to one and only one shift $s \in S$. Let $c_s \geq 0$ be the cost of shift s as described by the sum Equation (3.7). We can then describe the BDSP as a Set Partitioning Problem.

Set Partitioning Problem

Let L be a set, $\mathcal{P} \subseteq 2^L$ a set of subsets of L. Let $A = (a_{\ell s} : \ell \in L, s \in \mathcal{P})$ be the *incident matrix* defined, for every $\ell \in L$ and $s \in \mathcal{P}$, as follows.

$$a_{\ell s} = \begin{cases} 1 & \text{if shift } s \text{ covers bus leg } \ell, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\boldsymbol{c}$ be a cost vector $\boldsymbol{c} = (c_1, c_2, \ldots, c_{|P|}) \in \mathbb{R}^{|P|}$. The Set Partitioning Problem is the minimisation problem

$$\min \sum_{s \in \mathcal{P}} c_s \, x_s \tag{4.1a}$$

$$\text{s.t.} \sum_{s \in \mathcal{P}} a_{\ell s} \, x_s = 1 \quad \forall \ell \in L \tag{4.1b}$$

$$x_s \in \{0, 1\} \quad \forall s \in \mathcal{P}. \tag{4.1c}$$

Depending on the BDSP formulation, the master problem can be modelled as a *Set Covering Problem* [DS89]. The formulation is the same as the Set Partitioning Problem, but the equality in Equation (4.1b) is replaced by the inequality $\sum_{s \in \mathcal{P}} a_{\ell s} \, x_s \geq 1$.

Set Covering Problem

Let L be a set, $\mathcal{P} \subseteq 2^L$, a set of subsets of L. Let $A = (a_{\ell s} : \ell \in L, s \in \mathcal{P})$ the incident matrix and a vector $\boldsymbol{c} \in \mathbb{R}^{|S|}$. The Set Covering Problem is the minimisation problem

$$\min \sum_{s \in \mathcal{P}} c_s \, x_s \tag{4.2a}$$

$$\text{s.t.} \sum_{s \in \mathcal{P}} a_{\ell s} \, x_s \geq 1 \quad \forall \ell \in L \tag{4.2b}$$

$$x_s \in \{0, 1\} \quad \forall s \in S. \tag{4.2c}$$

In theory, if we could enumerate all shifts in $\mathcal{P}$, the optimal solution to the BDSP could be found by solving the Integer Linear Programming (ILP) model of the Set Partitioning Problem or Set Covering Problem optimally. However, this is unpracticable in most cases.

Anthony Wren modelled [SW88] the BDSP as a Set Covering Problem in 1988 and solved it using Branch and Bound. Other researchers used Column Generation with a Set Covering Equation (4.2) or Set Partitioning Equation (4.1) master problem and the Resource Constrained Shortest Path Problem (RCSPP) as subproblem [SW88, DS89, PLP08, LH16].

4.3 Heuristic and Hybrid methods

Exact algorithms methods are nowadays faster than they have ever been, and in most cases, they can provide a good solution in a reasonable time. Nevertheless, finding the optimal solution and proving optimality is impracticable in most real-life cases. Here, heuristic methods help us.

In the literature, many methods have been used, such as Greedy [MT86, TK13] Exhaustive Search [CSSC13], Tabu Search [LPP01, SK01b], Genetic Algorithms [LPP01, LK03], or Iterated Assignment heuristics [CdMdA+17].

However, most work focuses mainly on cost, rarely minimising idle time and vehicle changes [IRDGM15, CdMdA+17]. Break constraints are mostly simple, often including just one meal break. Complex break scheduling within shifts has been considered by authors in different contexts [BGM+10, WM14]. There is not much work on multi-objective bus driver scheduling [LPP01], but multi-objective approaches are used in other bus operation problems [RMVP13].

Regarding Hybrid Methods: In 2022, Rosati et al. used [RKB+23] the Set Partitioning Problem for the Solving phase in CMSA: a recently developed matheuristic. Their implementation of CMSA showed good performances on the large instances, which are, in general, the most critical ones.

In Figure 4.1, we show the ConnectedPapers graph of the paper [KMM22]; to generate it, we used the free tool available on the net[2].

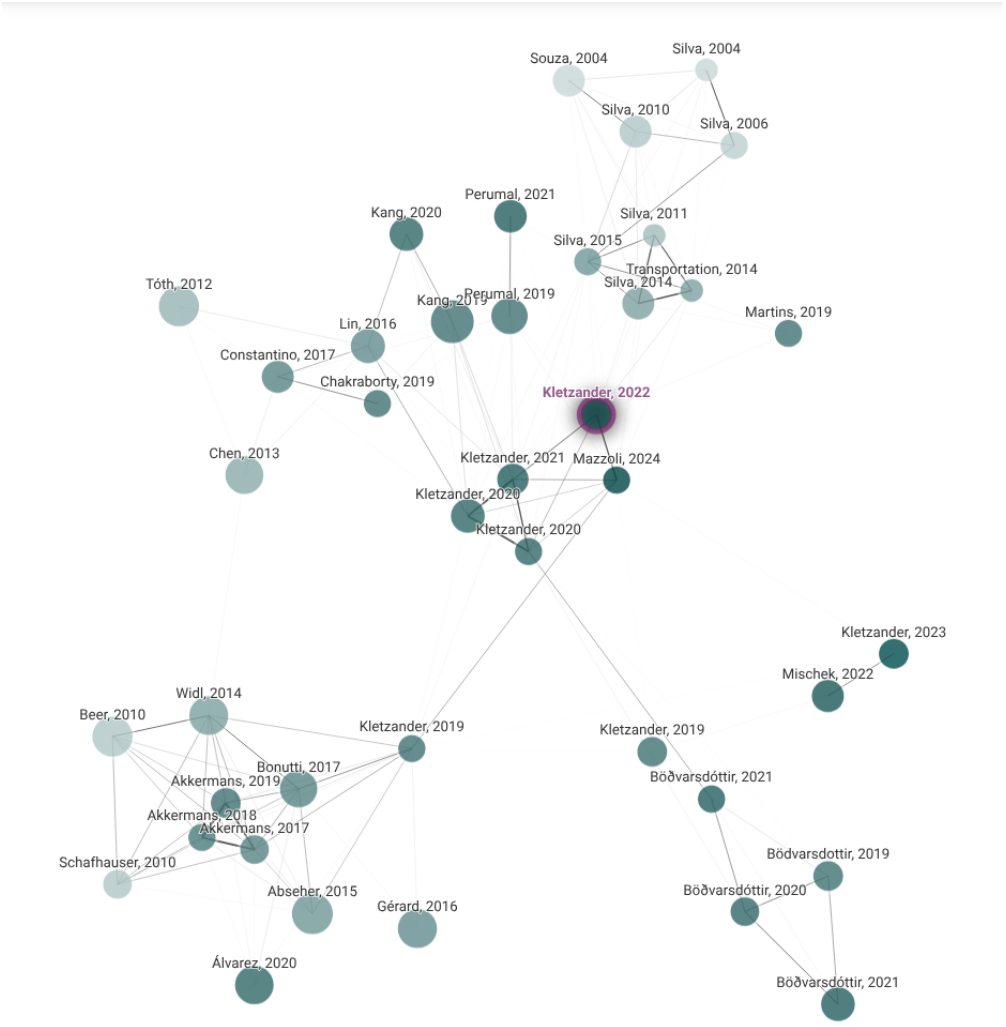


Figure 4.1: Graph from ConnectedPapers based on [KMM22]. Each node is an academic paper related to the original paper. Papers are arranged according to their similarity (this is not a citation tree). Node size is the number of citations. Similar papers have strong connecting lines and cluster together.

[2] https://www.connectedpapers.com/main/2c402f3d4d30fb3eea188e834ffba7ae3caf9692/graph?utm_source=share_popup&utm_medium=copy_link&utm_campaign=share_graph

CHAPTER 5

Metaheuristic methods for the Bus Driver Scheduling Problem

About this chapter

Many exact approaches for the Bus Driver Scheduling Problem have been developed. In the literature, Branch and Price (B&P) is the best-performing algorithm, optimally solving instances of up to 10 bus tours in few seconds. It is very effective for instances of up to 40 bus tours. However, for larger instances, B&P struggles to find good solutions in a reasonable time. This is why we need to rely on metaheuristics to get high-quality solutions. In this chapter, we propose metaheuristic algorithms to tackle large instances of the BDSP.

This approach was originally published in a conference paper at GECCO 2022 [KMM22].

5.1 Related work

Because of the need to solve very large real-world problems in a reasonable time, several heuristic methods have been studied for the Bus Driver Scheduling Problem (BDSP). Some examples are Greedy [MT86], Tabu Search [SK01a], Simulated Annealing [KM20], GRASP [DLFM11, PPÁME24], Guided Local Search [SS15], and Genetic Algorithm [LK03, LPP01].

However, the problem definition of the BDSP is highly dependent on the country's labour regulations, and algorithm performance can differ significantly on two different definitions. As described in Chapter 3, we focus on a variant of the BDSP that follows the Austrian rules, which are particularly strict regarding rest and driving breaks. This

specific set of rules are not found in the literature. Therefore, we cannot directly use previous algorithms for our variant.

Regarding the problem introduced in Chapter 3, to the best of our knowledge, the recently introduced approaches based on Branch and Price (B&P) [KMVH21], and Simulated Annealing (SA) [KM20] represent the current state of the art for this problem. Although these approaches achieve very good results, the optimal solutions are not yet known for large instances.

Heuristics and Metaheuristics The word *heuristic* comes from the Ancient Greek word ενρισκείν (*heuriskein*), which means *I find, I discover*. For a historical background, the reader can check the *Handbook of Metaheuristics* by Michel Gendreau and Jean-Yves Potvin [GP19, Chap. 2.3.1].

The term *metaheuristic* was introduced by Fred W. Glover (Figure 5.1) in 1986 [Glo86, p. 541] in the article *Future paths for integer programming and links to artificial intelligence*.[1] The Greek prefix *meta* means "upper-level methodology". Metaheuristics are solution methods that were designed to solve problems too difficult for exact algorithms. Hence, metaheuristics provide "acceptable" solutions in a reasonable time for solving hard and complex problems in science and engineering. This explains the significant growth of interest in the metaheuristic domain. Unlike exact optimisation algorithms, metaheuristics usually guarantee neither the optimality of the obtained solutions nor the convergence to optimal solutions.

Figure 5.1: Fred Glover and I at MIC 2024, Lorient (France)

We distinguish two types of metaheuristic methods:

Population-based They can be viewed as an iterative improvement in a population of solutions. At first, the population is initialised, then a new one is generated, and finally, these populations are integrated into a new one using a selection procedure.

Single-solution While solving optimisation problems, trajectory metaheuristics improve a single solution. They can be viewed as "walks" through neighbourhoods or search trajectories through the problem's search space.

[1] Glover uses the word metaheuristic only once in the whole document: «Tabu search may be viewed as a "meta-heuristic" superimposed on another heuristic».

5.2 Tabu Search

Tabu Search (TS)[2] is a method introduced by Fred W. Glover in his work *Future paths for integer programming and links to artificial intelligence* in 1986 [Glo86]:

> «Tabu search may be viewed as a "meta-heuristic" superimposed on another heuristic. The approach undertakes to transcend local optimality by a strategy of forbidding (or, more broadly, penalizing) certain moves. The purpose of classing a move forbidden — i.e. "tabu" — is chiefly to prevent cycling. In view of the mechanism adopted for this purpose,the approach might alternatively be called "weak inhibition" search, for the moves it holds tabu are generally a small fraction of those available, and a move loses its tabu status to become once again accessible after a relatively short time. (In this respect the method may be contrasted to branch and bound, which likewise forbids certain moves to prevent cycling, but in a more rigid fashion-a form of "strong inhibition" search.»

TS is one of the most studied and used metaheuristic algorithms for combinatorial optimisation. This method can provide good solutions very close to optimality. Therefore, these successes have made TS popular [GP19]. The main ideas of TS are:

1. To allow to move to solutions *worse* than the current one, if necessary.
2. To *forbid* certain moves in order to escape from local optima. This is achieved using a *tabu list*, which is an array that stores the performed moves of the algorithm.

The algorithm, based on the original version by Glover [Glo86, Sec. 5], is summarised in Algorithm 5.1.

Implementation of TS for the BDSP In our implementation, we use two different move types to generate the neighbourhood of the solution: `Shift` and `Swap`. To avoid confusion between shifts (a solution component) and shift moves, we refer to shifts as *employees* in this section. Consider a solution S; we define the following two moves:

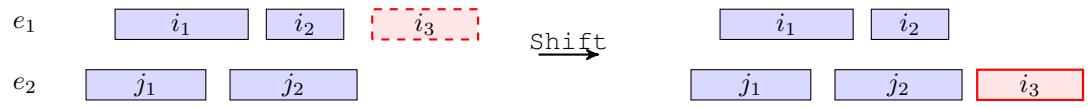
- `Shift`(e_1, i, e_2): Select two employees $e_1, e_2 \in S$ and a bus leg $\ell \in e_1$ uniformly at random. Then, remove bus leg ℓ from e_1 and assign it to e_2. Figure 5.2a shows an example of a `Shift` move.
- `Swap`(e_1, i, e_2, j): Select two employees $e_1, e_2 \in S$ and two bus legs $i \in e_1$ and $j \in e_2$ uniformly at random. Remove the bus leg i from e_1 and assign it to e_2; similarly, remove j from e_2 and assign it to e_1 (see Figure 5.2b).

[2]The name tabú comes from *tàpu*, a Polynesian word that Captain James Cook imported in 1777.

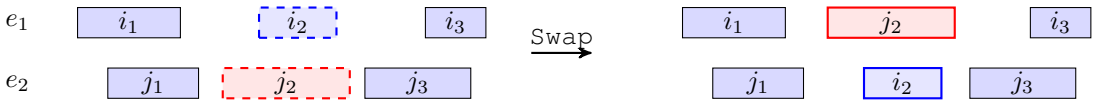
Algorithm 5.1: Tabu Search

Input: $t_{\max}$: max run time, S_0: initial solution, N: a neighbourhood
Output: S^*

Initialise tabu list T;
$S^* \leftarrow S_0$;
repeat
  Identify the best solution $y \in N(S^*)$ and the best non-tabu solution $y_{\text{nt}} \in N(S^*)$;
  if *Y is not tabu or (Y is tabu and* $z(Y) \leq \min\{z(S^*), z(Y_{\text{nt}})\}$*)* **then**
    $S^* \leftarrow Y$;
  else if Y_{nt} *exists* **then**
    $S^* \leftarrow Y_{\text{nt}}$;
  end
  else
    return S^* ; `// No improving move found`
  end
  Update tabu list T;
until *termination criterion is met*;
return S^*



(a) Shift move `Shift`(e_1, i_3, e_2)



(b) Swap move `Swap`(e_1, i_2, e_2, j_2)

Figure 5.2: Illustration of (a) `Shift` Move and (b) `Swap` Move.

Note that performing moves can lead to infeasible solutions (e.g., the new bus leg may overlap with another one). We penalise such solutions by adding a very high penalty hard-constraints-violation to the objective function, while specifically trying to repair an infeasible solution would be very challenging due to the complex constraints. For instance, assume that the employee e works on two bus legs ℓ_1, ℓ_2 at the same time for k minutes. Then, we set the violation of hard constraints to 1000[3].

Given a solution S, we define three neighbourhoods:

- $N_{\texttt{Shift}}(S)$ is the set of solutions that can can be reached from S performing a `Shift` move.
- $N_{\texttt{Swap}}(S)$ is the set of solutions that can can be reached from S performing a `Swap` move.
- $N_{\texttt{Combination}}(S)$ is the union set $N_{\texttt{Combination}}(S) := N_{\texttt{Shift}}(S) \cup N_{\texttt{Swap}}(S)$.

Now that we defined the neighbourhoods, we still have to describe how to explore them.

Initial Solution

The initial solutions were generated using a Greedy construction method [KM20]. The solution is generated by sequentially assigning bus legs to the employee where the lowest additional cost is incurred, or to a new employee if this would incur an extra cost of at most 500 compared to the best existing employee assignment.

Search Strategy

To explore the neighbourhood, we use the *First Improvement* strategy. The idea is to reach a local optimum quickly, by avoiding the complete exploration of the neighbourhood. The First Improvement technique starts from a solution S, evaluates the neighbouring candidate solutions in $N(S)$ and selects the first $S' \in N(S)$ for which $z(s') < z(S)$, that is, the first improving neighbour encountered. As soon as it finds it, it moves towards the new solution and assigns $S^* = s_1$. The algorithm starts again from S_1, exploring the neighbourhood $N(s_1)$. Therefore, after a finite number of iterations, First Improvement method returns a solution S^* that cannot be improved in its neighbourhood $N(S^*)$. This is a local optimum. As said in Lemma 1, the set of bus legs L is totally ordered. The order relationship can be extended to a order relationship of the employees: let $e_1, e_2 \in S$ be two employees and ℓ_1 and ℓ_2 be the first legs assigned to, respectively, e_1 and e_2. We say that $e_1 \preceq e_2$ if $\ell_1 \preceq \ell_2$; in other words, if e_1 starts to work before e_2.

In our implementation, *First Improvement* scans the neighbourhood according to the order of employees returned from the initial solution, and the order of legs according

[3]Basically, we change the objective function from $z(S)$ to $z(S) + 1000 \cdot h(S)$, where $h(S)$ is a measure of the hard-constraint violations of the solution S

to start time, and returns the first improving neighbour. We show First Improvement in Algorithm 5.2, using z as the objective function in Equation (3.7) including hard constraint penalisation.

Algorithm 5.2: First Improvement

Input: N: neighbourhood, S_0: initial solution
Output: S^*: local optimum

```
S* ← S_0;
for every Y in the neighbourhood N(S*) do
    if z(Y) < z(S*) then
        S* ← Y;
        break; // Stop after the first improving move
    end
end
return S*
```

The *tabu list* represents the core of Tabu Search. Let S be a solution; we represent the tabu list as an integer matrix $T = (t_{s,i}) \in \mathbb{Z}^{|S|\times|L|}$, where L is the set of bus legs. At the beginning of the run, the matrix is initialised assigning $t_{s,i} \leftarrow -\left\lceil \frac{\alpha}{10}\sqrt{|L|} \right\rceil$ for every $s \in S$, $i \in L$, where $\alpha \in \mathbb{R}_{\geq 0}$ is a parameter that has to be tuned.

Let us assume that we are at iteration k and perform a `Shift` move. We remove the bus leg i from employee e_1 and assign it to employee e_2. Then, the tabu list is updated, and the iteration in which the move is performed is stored: $t_{e_1,i} = k$. To check whether the move $\texttt{Shift}(e_1, i, e_2)$ is tabu, we check whether the current iteration k fulfils the following condition:

$$k - t_{e_2,i} < \left\lceil \frac{\alpha}{10}\sqrt{|L|} \right\rceil . \tag{5.1}$$

This condition checks if a sufficient number of iterations have passed, we do this to avoid cycling. We use $\sqrt{|L|}$ as a measure of the size of the problem (the square root is due to the large number of bus legs). If this condition is satisfied, then the move $\texttt{Shift}(e_1, i, e_2)$ is considered tabu.

As a swap can be seen as a composition of two `Shift` moves, the tabu status can be applied to those individual `Shift` moves. Therefore, a swap move is tabu if any of the two `Shift` moves is tabu. We allow a move (even a tabu one) if it improves the best solution so far. As termination criterion, we use the maximum runtime $t_{\max} = 3600\,\text{s}$ since it is the one used in the literature.

TS is deterministic, since all the components of the algorithms are themself deterministic.

5.3 Iterated Local Search

Iterated Local Search (ILS) [LMS19], is a non-deterministic metaheuristic algorithm. It is composed of three parts: LocalSearch, Perturbation, and AcceptanceCriterion. In the LocalSearch phase, we improve the current solution, reaching a local optimum. In the Perturbation phase the solution is perturbed using random moves. After that, a Local Search is performed again. After this process, we obtain two local optima, and we must choose which one we accept. This is done with the AcceptanceCriterion. The structure of this algorithm is sketched in Algorithm 5.3.

Algorithm 5.3: Iterated Local Search

Input: $t_{\max}$: max run time, S_0 : initial solution
Output: S^*

```
1 S* ← LocalSearch(S_0);
2 while termination criterion is not met do
3 |   S ← Perturbation(S*);
4 |   S' ← LocalSearch(S);
5 |   S* ← AcceptanceCriterion(S*, S');
6 end
7 return S*;
```

For the LocalSearch phase, we investigate two techniques, each with different neighbourhoods: `FirstImprovement` (described in Algorithm 5.2) and `TabuSearch` (Algorithm 5.1). In this case, `TabuSearch` is considered as a black-box algorithm that returns a solution; it terminates when a maximum number of iterations without improvement $k_{\max} \in \mathbb{N}$ is reached, this parameter has to be tuned.

In order to escape from a local optimum, we perturb the solution m times (where m is a parameter that we must tune). Let S be a solution. We can apply three different types of moves in Perturbation(S).

- `Shift`: like the neighbourhood `Shift` previously described.
- `Kick`: Select three employees $e_1, e_2, e_3 \in S$ and two bus legs $i \in e_1$, $j \in e_2$ uniformly at random. Then, remove bus leg i from e_1 and assign it to e_2, and remove bus leg j from e_2 and assign it to e_3.
- `Crossover`: Select two employees $e_1, e_2 \in S$ uniformly at random. Select t in a uniform way in the real interval $[t_{\min}, t_{\max}]$, where
  - $t_{\min}$ is the minimum of the start times of the two employees:

$$t_{\min} = \min \left\{ \min_{\ell \in e_1} \{start_\ell\}, \min_{\ell \in e_2} \{start_\ell\} \right\}.$$

– $t_{\max}$ is the maximum end time:

$$t_{\max} = \max\left\{\max_{\ell \in e_1}\{end_\ell\}, \max_{\ell \in e_2}\{end_\ell\}\right\}.$$

After that, swap every bus leg ℓ between e_1 and e_2 with $start_\ell \geq t$.

As acceptance criterion, we apply three possibilities, as described in [LMS19].

- `Better`(S, S'): Accept the new solution S' only if it improves the best one so far, i.e., if $z(S') < z(S)$.
- `RW`(S, S'): (Random Walk) Always accept the new solution S'.
- `LSMC`(S, S') (Large Step Markov Chain): This criterion comes from Simulated Annealing (SA). LSMC is somehow an intermediate condition between the first two. If the new solution S' is better than S, then accept it. Otherwise, accept it with probability $e^{\frac{z(S)-z(S')}{\tau}}$, where $\tau \in \mathbb{R}_{>0}$ is a parameter that has to be tuned. If τ is very large, then the criterion becomes similar to `RW`. If τ is very small, it becomes similar to `Better` criterion.

Finally, regarding the termination criterion, we use a maximum runtime $t_{\max} = 3600\,\text{s}$.

5.3.1 Experiment 1: Parameters Tuning

Research question

What is the best set of parameters for the ILS and TS?

Task

The parameters to be tuned are:

- For TS: the parameter α, as described in Equation (5.1). Instead of considering a generic $\alpha\sqrt{|L|}$, with $\alpha \in \mathbb{R}$, we use $\frac{\alpha}{10}$ with $\alpha \in \mathbb{N}$ to allow an easier tune of the parameter among a finite set.
- For ILS:
  - The number of maximal iterations without improvements $k_{\max}$.
  - The number of times m we apply Perturbation.
  - For the AcceptanceCriterion, the parameter τ of LSMC.

Setup

We used a computing cluster that has 10 nodes, each having 24 cores, with an Intel Xeon E5-2650 v4 2.20 GHz CPU and 252 GB RAM. Each run is executed in a single thread.

To tune the parameters, we used the automated tool *Sequential Model-Based Algorithm Configuration* (SMAC3) [LEF+22], version 1.1.1.

We split the available instances into a training set composed of 26 instances (40% of the instances) and a test set of 39 instances (60% of the instances). Each trial used the following options:

- Maximum amount of wallclock-time used for optimisation: 108 h.
- Maximum runtime, after which the target algorithm is cancelled: 1 h.
- As the metric quality we consider the *percentage of improvement* PI between the resulting solution S and the initial one S_0, i.e. $PI(S) = \frac{z(S_0)-z(S)}{z(S_0)} \cdot 100$. This value is either 0 (if the algorithm cannot find an improvement of the initial solution, or strictly positive (when it does find the improvement). We do this because SMAC3 only minimises the target function.
- For ILS we use the following options:
  - Neighbourhood: $N_{\texttt{Shift}}$
  - LocalSearch: `First Improvement`.
  - AcceptanceCriterion: `LSMC`.
  - Perturbation: `Kick`.

Results

The tuned values reported by SMAC3 are described in Table 5.1.

Table 5.1: Parameters tuned

Parameter	Range	Final value
$\alpha \in \mathbb{N}$	$\{1, \ldots, 20\}$	12
$m \in \mathbb{N}$	$\{1, \ldots, 50\}$	2
$k_{\max} \in \mathbb{N}$	$\{1, \ldots, 50\}$	48
$\tau \in \mathbb{R}$	$[100, 10\,000]$	2408.37

5.3.2 Experiment 2: Impact of Algorithmic components

Research Question

What impact do the algorithm components have on the algorithm?

Task

With these experiments, we want to get insights into the impact of the components of our heuristics on their performance. In order to do that, we ask the following questions:

- Regarding TS, how much does the choice of the neighbourhood impact on the results? We compare Tabu Search with $N_{\texttt{Shift}}$ and $N_{\texttt{Combination}}$ neighbourhoods. We use $N_{\texttt{Combination}}$, but not $N_{\texttt{Swap}}$, since $N_{\texttt{Swap}}$ does not allow to change the number of bus legs assigned to an employee on its own.
- Regarding ILS we wonder what the impact of the components Local Search, Perturbation, and AcceptanceCriterion is:
  - Fix ILS with LocalSearch=`TabuSearch` with $N_{\texttt{Shift}}$, and AcceptanceCriterion=`LSMC`. How does the choice of Perturbation among `Kick`,`Shift`,`Crossover` affects the performance?
  - Fix ILS with LocalSearch=`TabuSearch` with $N_{\texttt{Shift}}$, and Perturbation=`Shift`. How does the choice of AcceptanceCriterion among `LSMC`, `RW`, and `Better` affect the performance?
  - Fix ILS with AcceptanceCriterion=`RW`, and Perturbation=`Shift`. How does the choice of ILS' LocalSearch between `FirstImprovement` or `TabuSearch` relates with the performance ?

Setup

Regarding the hardware, we use the same setup as in Section 5.3.1. The Tabu Search Algorithm is deterministic: Repeated runs produce the same result. Instead, Iterated Local Search is non-deterministic, therefore, we execute 10 independent runs for every instance.

We used the publicly available set of 50 benchmark instances provided by [KM20][4] together with the new 15 instances described in Section 3.3.

As quality metric, we evaluate the GAP from the known-best-solution in the literature. For the larger new instances, we set S_{bks} as the best results from our algorithms.

$$\text{GAP}(S) = \frac{z(S) - z(S_{\text{bks}})}{z(S_{\text{bks}})} \cdot 100. \tag{5.2}$$

We tested Iterated Local Search with the following options:

- Neighbourhoods: $N_{\texttt{Shift}}$,
- LocalSearch: `FirstImprovement` and `TabuSearch`

[4] https://cdlab-artis.dbai.tuwien.ac.at/papers/sa-bds/

- Acceptance Criterion: `RW`, `Better`, `LSMC`
- Perturbation: `Kick`, `Shift`, `Crossover`

As determined by SMAC3, two perturbations are executed at every iteration. We tested the perturbations by fixing the other components to Tabu Search with $N_{\texttt{Shift}}$ and `LSMC`. We tested two different Local Search algorithms while `RW` was used as Acceptance Criterion and `Shift` as perturbation: Tabu Search and First Improvement. First Improvement is simpler, and it only focuses on intensification (no diversification is done). Instead, Tabu Search is a more complex algorithm and it also has a diversification phase. As figure 5.5c shows us, , and performs better than First Improvement.

Results/Visualisation

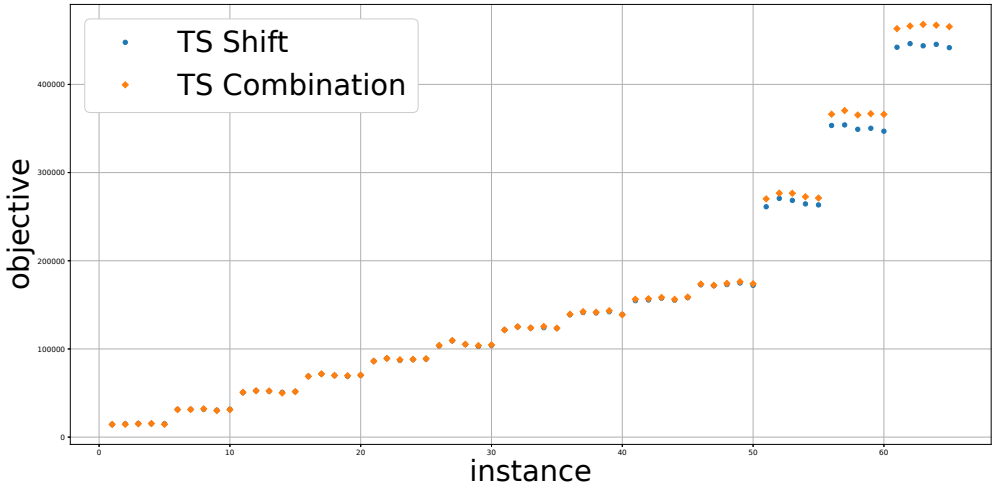


Figure 5.3: Comparison of Tabu Search neighbourhoods: Objective function values

Observations

Regarding TS, Figure 5.3 shows us the benefits of using the neighbourhood $N_{\texttt{Shift}}$ over $N_{\texttt{combination}}$ for instances of size $150, 200, 250$ described in Section 3.3. Comparing the GAP across all the 65 instances, Figure 5.4 confirms the result. Note that the minimum in the case of $N_{\texttt{Shift}}$ neighbourhood is negative because the algorithm could achieve a new solution for the instance `realistic_100_48`.

Regarding ILS, Figure 5.5 shows the GAP across the 65 instances for the three components. In Figure 5.5a we note that the GAP with `Crossover` is higher, with respect to the other two. Even if `Kick` and `Shift` seem to perform in a similar manner, we can see that `Shift` has a lower median and first quartile. Finally, in Figure 5.5c we observe that the variant with `TabuSearch` has lower minimum, median, and third quartile than `FirstImprovement`.

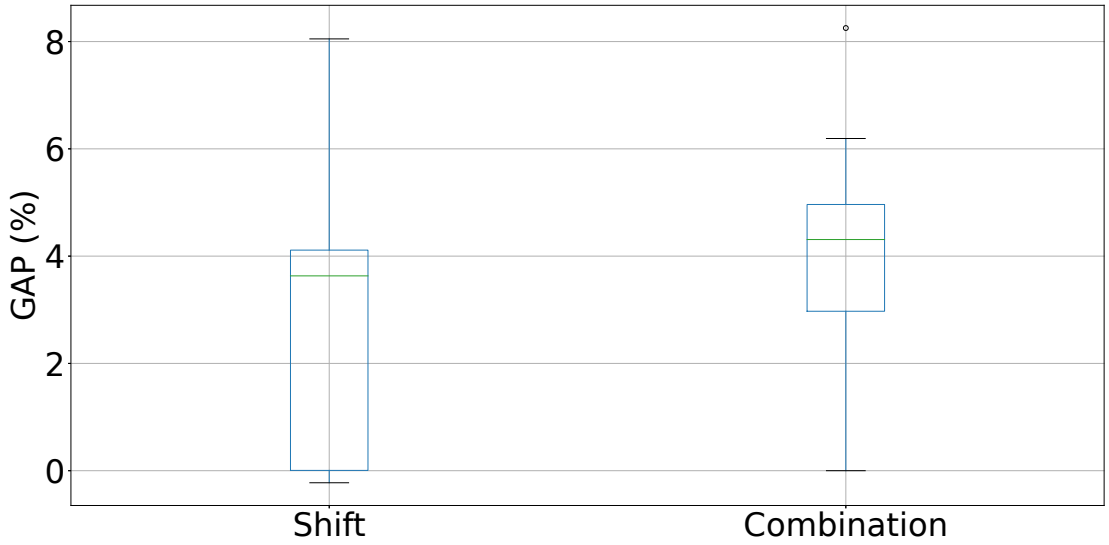


Figure 5.4: Comparison of Tabu Search neighbourhoods: GAP comparison

Discussions

We chose $N_{\texttt{Shift}}$ as the best version of TS for the remaining evaluations. This is because of its better performance, probably due to the fact that the neighbourhood $N_{\texttt{Combination}}$ is much larger than $N_{\texttt{Shift}}$.

For ILS, we decide to use `Shift` as Perturbation. We also see that there is no AcceptanceCriterion that dominates all the others. However, for 10 instance groups out of 13, `RW` performs a bit better than the other two. For LocalSearch, we observed that `FirstImprovement` performs worse than `TabuSearch`, probably because the first is simpler and it only focuses on intensification (it does no diversification), while `TabuSearch` is a more complex algorithm and it also has a diversification phase. In summary, we consider the version of Iterated Local Search with Perturbation = `Shift`, AcceptanceCriterion = `RW`, and LocalSearch = `TabuSearch`.

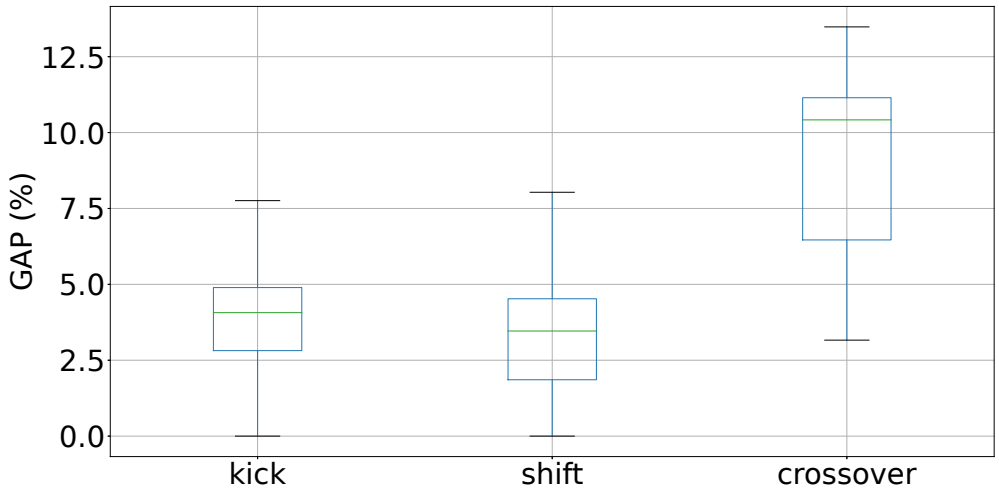
5.3.3 Experiment 3: Comparison with the literature

Research Question

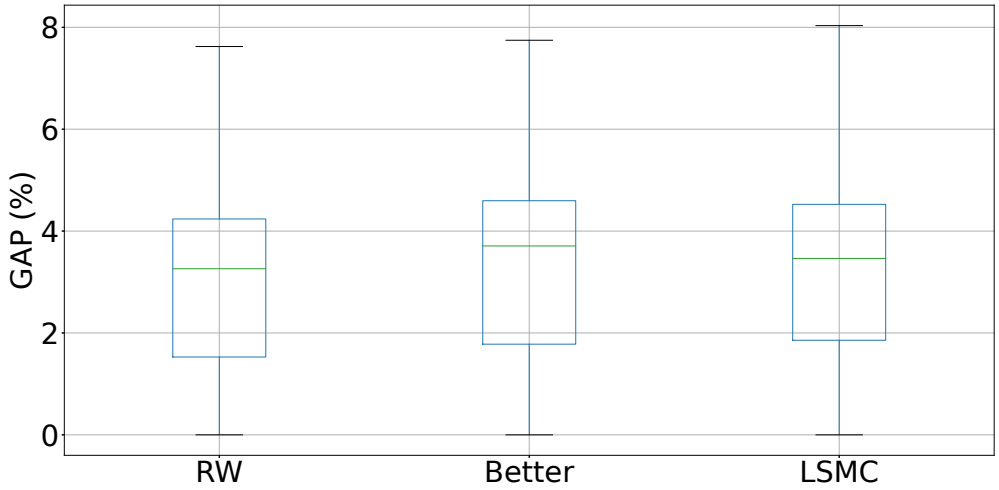
How do our selected version of Tabu Search and Iterated Local Search compare with the ones in the literature?

Task

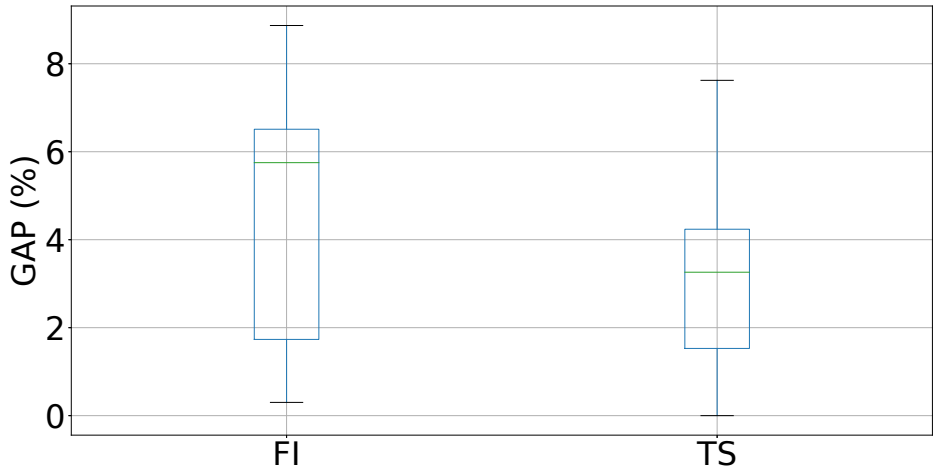
We want to compare the objective function values of our best versions of Tabu Search and Iterated Local Search with the one of other algorithms in the literature: Simulated Annealing [KM20], Hill Climbing [KM20], and Branch and Price [KMVH21].



(a) Comparison of Perturbation for ILS. Here we fix LocalSearch=TS and AcceptanceCriterion=LSMC



(b) Comparison of AcceptanceCriterion for ILS. Here we fix LocalSearch=TS and Perturbation=Shift



(c) Comparison of LocalSearch for ILS. Here we fix AcceptanceCriterion=RW and Perturbation=Shift

Figure 5.5: Comparison of the three components of Iterated Local Search

We also performed statistical tests. First, we fixed a significance level of 0.05. Then, as described by Calvo [CS16], we first performed the Friedman test to detect whether all algorithms perform the same. Formally, this means having a null hypothesis H_0 : the performance of all the algorithms is the same. This hypothesis is rejected with a p-value smaller than $2.2\,\mathrm{e}^{-16}$. Finally, we compared multiple algorithms using the Nemenyi *post-hoc* test [CS16].

Setup

We use the same setup as in Section 5.3.1. Note that SA and Hill Climbing (HC) were executed in the same way as in their original publication.

For the statistical tests, we use the R script scmamp[5] [CS16].

Results/Visualisation

Table 5.2 shows the results for our final configurations in comparison with the literature. Each row shows the average results over the five instances of the respective size category. The best metaheuristic results are presented in **bold**; the lowest overall results are in *italics*.

Since larger instances are of particular interest for metaheuristics, Table 5.3 presents a detailed comparison for size 100.

Table 5.4 shows the results for the fifteen instances with size 150, 200, and 250. The results of these sizes do not include Branch and Price, because it was not able to provide a feasible solution after 1 h of runtime.

Regarding the statistical tests, we graphically show in Figure 5.6 the results on all instances with a *critical difference* (CD) plot. For ILS the LocalSearch, AcceptanceCriterion, and Perturbation are distinguished. Each considered algorithm is placed on the horizontal axis according to its average ranking for the instances. The performances of those algorithm variants below the critical difference threshold (1.8787) are considered statistically equivalent. In the CD plot, this is remarked by a horizontal bar that joins different algorithms.

Observation

Table 5.2 shows that Branch and Price achieves the best solutions for most instances across all size categories. As regards the metaheuristics, Simulated Annealing (SA) provides very good results for the smaller categories. However, these are precisely the categories where B&P is most useful, limiting the overall utility of this method. HC performs best in some mid-sized and the largest category, making it still a good choice. We note that TS has the best results for sizes 80 and 90 and a very close result for 100. In comparison, ILS can outperform TS in the smaller categories. However, on the larger

[5] https://github.com/b0rxa/scmamp/tree/e435f9d48078f93ab49b23a19fdb6ef6e12ea5f9

Table 5.2: Results for the benchmark instances grouped by size. The time values are in seconds.

Size	BP		SA			HC			TS	ILS		
	time	best	time	best	mean	time	best	mean		best	mean	std
10	7.2	*14 709.2*	22.8	14 717.4	**14 739.6**	7.8	14 904.4	14 988.4	15 036.4	14 832.2	14 900.0	65.2
20	1201.4	*30 294.8*	62.2	30 860.6	**30 970.8**	28.0	30 931.4	31 275.6	31 248.4	30 921.4	31 158.0	133.3
30	3610.6	*49 846.4*	108.8	50 947.4	**51 257.8**	99.4	51 544.2	51 917.3	51 483.0	51 323.8	51 603.0	181.3
40	3605.8	*67 000.4*	267.0	69 119.8	**69 379.9**	151.2	69 533.6	71 337.6	69 941.2	69 848.6	70 238.6	237.6
50	3674.4	*84 341.0*	329.0	87 013.2	87 557.3	295.4	86 718.6	**87 262.5**	87 850.6	87 879.8	88 130.1	164.9
60	4373.2	*99 727.0*	543.6	103 967.6	104 333.1	432.8	103 780.0	**104 296.2**	104 926.2	104 970.4	105 440.1	229.8
70	6460.4	*118 524.2*	751.4	122 753.6	**123 225.6**	718.6	122 912.8	123 303.8	123 632.2	123 601.8	123 976.2	223.3
80	5912.4	*134 513.8*	1140.2	140 482.4	140 913.8	959.4	139 765.2	140 508.0	**140 482.4**	141 031.4	141 250.3	105.1
90	7390.4	*150 370.8*	1453.0	156 385.0	157 426.0	1516.6	156 239.4	156 862.5	**156 296.4**	156 831.2	157 216.7	229.5
100	7395.8	*172 582.2*	1449.4	173 524.0	174 501.6	1483.2	172 327.8	**172 909.0**	172 916.0	173 378.6	173 587.4	187.0

Table 5.3: Computational results for instances of size 100. The time values are in seconds.

Instance	BP		SA			HC			TS	ILS		
	time	best	time	best	mean	time	best	mean		best	mean	std
46	7814	184 186	2220	172 473	173 000.0	1525	*170 866*	**171 822.3**	172 798	172 764	173 010.5	142.0
47	7293	*165 268*	1219	172 589	173 972.5	1197	170 942	**171 365.7**	171 763	171 710	171 907.9	481.9
48	7259	176 430	1324	173 431	174 476.0	1911	173 722	174 118.3	***173 041***	173 852	174 027.8	62.6
49	7302	*171 275*	1062	176 083	176 833.7	871	174 490	175 060.7	**174 793**	175 938	176 158.0	122.0
50	7311	*165 752*	1422	173 044	174 225.7	1912	171 619	**172 178.0**	172 185	172 629	172 833.0	126.4

Table 5.4: Results for larger instances.

Instance	SA			HC			TS	ILS		
	best	mean	std	best	mean	std		best	mean	std
150_51	261 082	261 785.7	644.1	260 893	261 427.0	614.1	**261 246**	261 992	262 516.6	473.1
150_52	267 986	269 166.3	1139.5	267 265	**268 214.7**	1075.2	270 737	272 138	272 212.3	90.8
150_53	267 724	269 533.0	2871.8	265 820	**266 537.3**	696.1	268 473	269 324	269 497.0	124.2
150_54	265 132	267 243.3	1882.1	265 564	267 170.0	1531.0	**264 492**	266 388	267 128.9	463.6
150_55	264 922	265 799.3	1181.7	262 110	264 112.3	1904.9	**263 326**	264 625	264 811.1	180.2
150 mean	265 369.2	266 705.7	1543.8	264 330.4	**265 492.3**	1164.2	265 654.8	266 893.4	267 233.2	266.4
200_56	356 068	357 349.0	1704.1	355 143	356 350.3	1135.9	**353 411**	354 066	354 409.7	400.1
200_57	355 322	356 463.7	1453.6	355 519	356 878.3	1860.7	**354 059**	355 229	355 699.1	457.6
200_58	349 462	351 489.7	1807.5	350 878	351 275.0	653.6	**349 109**	349 143	349 895.4	590.2
200_59	353 838	354 732.7	775.0	350 708	351 830.0	975.5	**350 231**	350 377	350 637.4	220.0
200_60	350 309	352 006.7	1518.9	350 466	351 141.0	1133.8	**346 926**	347 648	347 958.2	270.0
200 mean	352 999.8	354 408.3	1451.8	352 542.8	353 494.9	1151.9	**350 747.2**	351 292.6	351 720.0	387.6
250_61	444 123	444 476.7	317.2	443 061	444 935.3	1970.1	**442 126**	448 881	449 633.8	471.1
250_62	447 642	448 627.7	905.3	446 834	448 376.3	2112.4	**446 221**	453 675	454 476.1	663.1
250_63	446 517	447 396.0	768.7	447 602	447 924.0	555.1	**443 777**	453 589	454 445.0	872.1
250_64	447 134	448 542.7	1753.1	446 314	447 239.3	872.9	**445 436**	453 133	455 126.3	4063.9
250_65	440 796	443 582.0	2000.8	440 496	**441 529.7**	897.7	441 669	451 945	454 053.0	4725.4
250 mean	445 242.4	446 525.0	1149.0	444 861.4	446 000.9	1281.6	**443 845.8**	452 244.6	453 546.8	2164.5

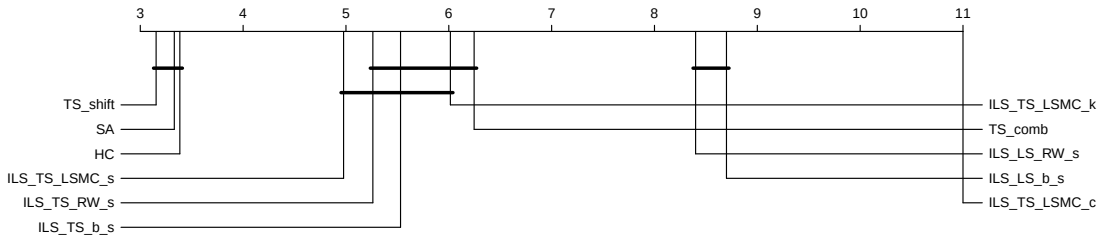


Figure 5.6: Critical difference plot for all the 65 instances. Groups of algorithms that are not significantly different (at $p = 0.05$) are connected.

ones, TS is clearly the better choice. ILS still reaches better averages than Simulated Annealing in the two largest categories but stays behind HC and TS.

Table 5.3 shows that B&P starts to fall behind, enabling a new best known solution to be found by TS.

Table 5.4 shows the results for the new larger instances, providing a new benchmark for metaheuristics to compete on.

Figure 5.6 shows that TS, SA, and HC perform significantly better than the other algorithms, whith no significant difference among them. However, when this test is done only for the new larger instances, TS is the leading method with significant difference to SA and HC, reinforcing the conclusion that TS is the best metaheuristic to use for larger instances of the BDSP.

Discussion

Overall, these results showed that TS can scale very well to even this very large scale, outperforming the previous state-of-the-art methods as well as ILS on most instances, allowing the optimisation of whole city networks in practical applications.

A further advantage is that TS is a deterministic algorithm, eliminating the spread of results found in the other metaheuristic methods. However, a non-deterministic implementation would be possible by using randomness for tie-breaking. The previous algorithms also had a maximum runtime of one hour (on a machine which was around 5% slower than ours according to a benchmark script). However, they were designed to stop when no improvement is found for a certain number of iterations. While we currently did not use such a termination criterion, mainly to compare our different algorithms configurations, TS could easily be fitted with such a criterion for shorter runtimes.

Even if B&P provides the best solutions for most instances across all size categories, it has disadvantages when it comes to scaling. First, even though these experiments were performed on a slightly faster machine than ours (of around 6 % according to benchmark scripts), much more than one hour of runtime was used for the larger categories. Second, already for category 100, for some of the instances, only solutions worse than those from

the metaheuristics could be found, indicating that further scaling would be problematic. Indeed, no competitive results could be obtained in a reasonable runtime when trying the approach on our new, even larger instances (the new categories progress in much larger size steps than before). This highlights that metaheuristics are still needed when dealing with larger instances.

5.4 Conclusions

In this chapter, we analysed two solution methods for the Bus Driver Scheduling Problem based on Tabu Search and Iterated Local Search. We thoroughly evaluated and compared different algorithmic components. Our experimental results show that especially Tabu Search can scale very well to larger instances out of reach for exact methods. In larger instances of sizes 150, 200 and 250, the state-of-the-art Branch and Price method fails due to a memory error.

In the next chapter, we will investigate and explore a hybridisation between heuristics and exact method.

CHAPTER 6

Large Neighbourhood Search for the BDSP

About this chapter

This chapter presents a Large Neighbourhood Search (LNS) approach for solving the Bus Driver Scheduling Problem (BDSP). We propose several novel destroy operators and an approach using Column Generation to repair the subproblem. We analyse the impact of the destroy and repair operators and investigate various possibilities to select them, including adaptivity. We further propose and evaluate a deeper integration of B&P and LNS, storing the generated columns from the LNS subproblems and reusing them for other subproblems, or to find better global solutions. We analyse the impact of this new strategy, and we observe that it improves several best-knownsolutions even more than B&P or LNS on their own, constituting a new state of the art for this problem.

This approach was originally published in a conference paper at ICAPS2024 [MKHM24].

6.1 Large Neighbourhood Search

The *Large Neighbourhood Search* (LNS) algorithm was introduced by Paul Shaw in 1998 [Sha98]. The algorithm starts from an already feasible solution. The main idea is to destroy part of a solution in order to obtain a subproblem that is easy to solve optimally or at least close to optimality. Selecting the part to destroy is done by a set Ω of *destroy operators* (also called *destroyers*), the operator to apply is chosen randomly proportional to a given weight vector $\boldsymbol{\rho}$. Solving the subproblem is done by a *repair operator*, often an exact method. We accept the new solution S' if $z(S') < z(S_{\text{bsf}})$, where z represents the

objective function described in Equation (3.7). The term S_{bsf} is the best-so-far solution. We show the pseudocode of LNS in Algorithm 6.1.

Algorithm 6.1: Large Neighbourhood Search

Input: k_0 (initial destruction size)
Output: S_{bsf} (best solution found)

```
k ← k_0;
Construct the initial solution S_0 using the Greedy algorithm;
S_bsf ← S_0;
Initialise the weights ρ;
while time < t_max do
    Select destroy operator ω ∈ Ω using ρ;
    S' ← r(ω(S_bsf, k));
    if z(S') < z(S_bsf) then
        S_bsf ← S';
    end
    Update the subproblem size k;
end
return S_bsf;
```

6.1.1 Destroy Operators

Since our repair mechanism can only produce complete shifts, the aim of the destroyer algorithms is to select a subset of shift $E' \subseteq E$ that is removed from the current solution. The size of the subproblem $k = |E'|$ is given to the destroy operator. We propose three distinct ways to select E':

Employees uniform (ω_{EW}): Select k employees uniformly at random.

Employees weighted (ω_{EW}): Select $\left\lfloor \frac{k}{2} \right\rfloor$ employees using their cost as weight, the others uniformly at random. This is motivated by the fact that employees with high cost have a higher potential to benefit from reoptimisation. The split is done since a combination of high-cost and low-cost shifts can have a better potential to balance the shifts in the subproblem, e.g., by transferring some legs from the high-cost shift to an underutilised shift.

Tour remover (ω_{TR}): A tour is uniformly selected and all employees that share at least one leg of this tour are removed. This process is iterated until at least k employees are removed. This operator is based on the idea of selecting employees that have something in common and therefore have a higher potential that useful recombinations of their shifts are possible, e.g., optimising when and where a bus is handed over from one driver to the other. Note that this operator might select more than k employees because it removes all the employees who share a tour.

However, tours are usually not shared by too many employees since this incurs extra cost, so $|E'|$ does typically not exceed k by much.

6.1.2 Repair Operators

Once a set of removed employees E' is selected, the repair mechanism needs to solve the subinstance that is created by using all legs ℓ assigned to any employee $e \in E'$ together with the immutable data for the whole instance. This subinstance represents a complete instance of BDSP and can therefore be solved with any solution method of choice.

Since the Branch and Price approach [KMVH21] is the most powerful for small instances (it can provide an optimal solution for instances with 10 tours within seconds), it is the best fit for solving these subinstances.

Branch and Price [BJN+98] works by splitting the problem into a master problem and a subproblem. The goal of the master problem is to select a subset of the shift set $\mathcal{P}$, such that each bus leg is covered by exactly one shift while minimizing total cost. This corresponds to the set partitioning problem described in Equations (4.1a) to (4.1c).

$$\text{minimise} \sum_{s \in \mathcal{P}} c_s \cdot x_s \tag{6.1}$$

$$\text{subject to} \sum_{s \in \mathcal{P}} a_{s\ell} \cdot x_s = 1 \qquad \forall \ell \in L \tag{6.2}$$

$$x_s \in \{0, 1\} \qquad \forall s \in \mathcal{P} \tag{6.3}$$

Here x_s is the variable for the selection of shift s. The objective Equation (6.1) minimizes the total cost, Equation (6.2) states that each bus leg needs to be covered exactly once (using $a_{s\ell} \in \{0, 1\}$ to indicate whether shift s covers leg ℓ), and Equation (6.3) states the integrality constraint. This constraint is relaxed to $0 \leq x_s \leq 1$ for the relaxed master problem which is repeatedly solved at each node of the branching tree. Instead of the full set of possible shifts $\mathcal{P}$, a subset *columns* is maintained by the algorithm. Once no more new shifts can be found by the subproblem, the result of the relaxed master problem provides a local lower bound for the solution of the integer problem.

The subproblem is a Resource Constrained Shortest Path Problem (RCSPP) [ID05] where each leg is represented by a node in an acyclic graph, and each possible shift corresponds to a path in this graph from a source node to a target node. Costs and constraints are represented by resources that are tracked for each path through the graph and need to adhere to certain limits. Duals from solving the relaxed master problem (no integrality constraint) are added for each node, and each resource-feasible path where the cost of the edges and nodes on the path minus the sum of all duals along the path is negative (negative reduced cost) have the potential to improve the solution of the master problem. The complex rules for each shift are modelled in this subproblem, making it very challenging to solve. Therefore, several optimisations were necessary to solve it efficiently [KMVH21].

Master problem and subproblem are repeatedly solved until no more path with negative reduced cost can be found. This part of the process is called Column Generation and results in the optimal solution for the relaxed master problem, however this result is usually fractional. Therefore, branching is done and Column Generation is repeated on a modified problem where some connections from the graph are removed. This branching process is repeated until all branches are closed or until timeout.

However, previous work [KMVH21] already shows that, for instances up to 60 tours, the results are very close to the optimum when solving Column Generation only on the root node and then solving the master problem with integrality constraint on the set of columns obtained during Column Generation. These solutions are often much faster, but achieve a GAP of around 1% while the following branching process only closes this remaining GAP very slowly.

Therefore, we propose to drop the aim of optimally solving the subinstance with Branch and Price, and instead only use Column Generation on the root node to get very good solutions to the subinstance very fast. In the evaluation, we compare using Column Generation (`CG`) with using full Branch and Price (`BP`).

Once the repair mechanism returns a solution consisting of employees E^* that contain all bus legs from E', the new solution for the full problem is provided by $(E \backslash E') \cup E^*$.

6.1.3 Subproblem size

An important parameter for Large Neighbourhood Search is the size of the subproblem. However, the appropriate size depends on the destroy and repair operators. In the case of our system, the destroy operators are easy and fast to apply, but the complexity of Branch and Price increases rapidly with the size. Even when just applying Column Generation, the size of the RCSPP in the subproblem still leads to considerable increases in runtime.

Therefore, based on preliminary experiments, the smallest subproblem size in use is $k_0 = 5$. This size can still be solved in a few seconds, so it is fast enough, but it also leads to a high number of improvements, so it is large enough to allow meaningful changes of the solution. In the process of the search this size can be increased if too many iterations without improvement occur. This indicates that a larger size might be needed to escape local optima.

We use a maximum size of $k_{\max} = 20$ since runtime grows rapidly and for larger size too much time would be spent on each individual subproblem. When running the algorithm, the size starts with an initial value of k_0, and is increased by 1 until reaching $k_{\max} = 20$ whenever the previous improvement was more than $n_{\max}$ iterations ago. As soon as an improvement is found, k is set back to the initial value k_0.

Algorithm 6.2: Adaptive Large Neighbourhood Search

Input: k_0 (initial destruction size)
Output: S_{bsf} (best solution found)

$k \leftarrow k_0$;
Construct the initial solution S_0 using the Greedy algorithm;
$S_{\text{bsf}} \leftarrow S_0$;
Initialise the weights $\boldsymbol{\rho}$;
while *time* $< t_{\max}$ **do**
    Select destroy operator $\omega \in \Omega$ using $\boldsymbol{\rho}$;
    $S' \leftarrow r(\omega(S_{\text{bsf}}, k))$;
    if $z(S') < z(S_{\text{bsf}})$ **then**
        $S_{\text{bsf}} \leftarrow S'$;
    end
    Update weights $\boldsymbol{\rho}$ and subproblem size k;
end
return S_{bsf};

6.1.4 Adaptivity

Adaptive Large Neighbourhood Search (ALNS) is an extension of LNS, where the weights $\boldsymbol{\rho}$ for selecting the operators are adapted dynamically based on their performance [RP06].

Our method takes into account the score and the time required by destroy operator i. At first, every component of the weight vector $\boldsymbol{\rho}$ is set to $\frac{1}{|\Omega|}$. The destroy operator is selected in a random way with weights $\boldsymbol{\rho}$ using the *roulette wheel principle*:

$$\mathbb{P}\left(i\text{-th operator is selected}\right) = \frac{\rho_i}{\sum_{j=1}^{|\Omega|} \rho_j} \tag{6.4}$$

The selected destroy operator is then applied to the current solution S, which results in a subproblem that is passed to the repair operator r. We update the weights considering the number of successes and the total time of its selections. A similar approach was used in a related crew scheduling domain [CMS19]. At iteration n, we update the weight ρ_i^n of the i-th destroy operator using the following equation:

$$\rho_i^{n+1} = \lambda \rho_i^n + (1 - \lambda) \frac{\sum_{j=0}^{n} \sigma_i^j}{\sum_{j=0}^{n} \tau_i^j} \tag{6.5}$$

where $\sigma_i^j = 1$ if the i-th operator has improved the best-known-solution at iteration j, otherwise 0. The denominator is a sum of runtimes, so τ_i^j represents the time the i-th operator took for the entire process (destroying + repairing) at iteration j. If operator i was not selected at iteration j, then $\sigma_i^j = \tau_i^j = 0$. The real parameter $\lambda \in [0, 1]$ controls the sensitivity of the weights. A value of λ close to 0 implies that the operator performance during the search has a large influence while a value of 1 keeps the initial

weights static. As long as the denominator is 0, the value of the fraction is set to 0. In this case, $\rho_i^{n+1} = \lambda \rho_i^n$.

Algorithm 6.2 presents the pseudocode of Adaptive Large Neighbourhood Search (ALNS).

6.2 Integration of Column Generation and Large Neighbourhood Search

When applying Large Neighbourhood Search (LNS) on an optimisation problem, by default the repair operators are used in a black-box fashion: The current subproblem is fed into the operator, the corresponding solution is used to update the solution to the overall problem, and it does not matter how it was obtained. Each subproblem is solved from scratch, and all additional information gained while solving the subproblem is discarded every time.

However, this might actually be inefficient, as information from solving each subproblem might be useful for future subproblems, or it might be used beyond individual subproblems to globally enhance the best solution. In this section, we present two novel integration techniques for combining LNS with a Column Generation (CG) repair operator that are generally applicable for this combination of solution methods.

6.2.1 Column Storage

The first integration is dedicated to the reuse of information between subproblems. Recall that each shift (column) s generated by CG is a subset of the bus legs $s \subseteq L$. Each time a subproblem i consisting of legs $L_i \subseteq L$ is solved, a large set of columns S_i is generated, and a solution to the subproblem $S_i^* \subseteq S_i$ is returned.

By default, the columns are regenerated for each subproblem, however, the same columns might be generated repeatedly by multiple subproblems. Therefore, for a potential improvement, a column storage $\hat{\mathcal{P}}$ is introduced, and after solving a subproblem, the update in Equation (6.6) is performed.

$$\hat{\mathcal{P}} \leftarrow \hat{\mathcal{P}} \cup \texttt{select}(S_i) \tag{6.6}$$

A subset of the newly generated columns is added to the column storage $\hat{\mathcal{P}}$, where the selection criteria can be chosen freely, including no storage, or storing all columns.

Now, each time a subproblem i is solved, the set of columns S_i can be initialized as seen in Equation (6.7).

$$S_i \leftarrow \{s \in \hat{\mathcal{P}} \mid \forall \ell \in s : \ \ell \in L_i\} \tag{6.7}$$

Each column from the storage that only contains legs that are part of the subproblem are added to the initial set of columns, therefore, these columns do not need to be rediscovered

again in the current subproblem. Using this column reuse strategy is denoted by adding +R to the LNS version resulting in the method called LNS+R(B).

6.2.2 Global Background Solver

The second improvement adds the global view of B&P into the local view based on subproblems in LNS. The whole set of columns $\hat{\mathcal{P}}$ can be used in a global master problem. Since in LNS we do not aim to solve the problem exactly, there is no need to solve the relaxed master problem, instead the integer problem can be solved to get the best possible solution for the current set of stored columns.

However, solving increasingly larger integer master problems takes time and memory. Therefore, we propose to use a second thread for this improvement, using the first thread entirely for LNS, while the background thread repeatedly solves the master problem for $\hat{\mathcal{P}}$. At the end of every repairing phase of LNS, the algorithms checks whether the solution from the second thread is better than the current best:

$$S_{\text{bsf}} \leftarrow \text{argmin}\left(z(S'), z(S_{\text{bsf}}), z(S_{\text{bg}})\right).$$

where S_{bg} is the solution from the second thread, S_{bsf} is the best-so-far and S' is the solution after the repairing phase, as described in Algorithm 6.2 . If S_{bg} is better, it replaces S_{bsf}, and LNS continues from this improved solution. This is only a mild form of parallelization that is easily applicable with current multi-core machines, but can be very beneficial to further improve the joint performance of the methods.

As the background solver repeatedly solves similar problems, and new columns are frequently added, two more considerations are relevant. First, columns for this solver are never removed, but only added, allowing a warm start from the previous result in each solving cycle. Second, the master problem might not be solved to optimality, but stopped according to a given criterion, since incorporating new columns might be more beneficial than spending more time on the current cycle. We propose to use a timeout t_{bg}, but at least solve the root node of the MIP, before ending the current cycle. Adding the background solver is denoted as +B for the LNS version.

This improvement using the background solver can also be used for Branch and Price on its own. By default, the integer master problem is solved there whenever Column Generation on a node is finished, or in case even the first node runs into timeout. However, as each integer solution is a global upper bound independent from the current position in the branching tree, this process can be done in the background solver as well on the set of columns S. We call this version BP+B in the evaluation.

6.2.3 Selection of Columns to Store

A critical parameter for the proposed improvements is the selection of columns to store via the `select` function. The easy way is to store all new columns, we refer to this option as (F) (full set). However, this might use a considerable amount of memory.

Further, the time to search for useful columns for the current subproblem might counter the benefit of not having to rediscover them, and the background master problem might get excessively large.

A very lightweight alternative would be to only select the best subset S_i^* for each subproblem. We denote this version as (B) (best). The focus on only the best columns will keep the size of $\hat{\mathcal{P}}$ low, but ideally still preserves very good solutions, while the selection of different subproblems over time should still provide some diversity.

Table 6.1 summarises the variants of LNS related to this section.

Table 6.1: LNS variants

Name	Column Reuse	Background Solver	`select`
LNS	no	no	-
LNS+R(B)	yes	no	best
LNS+R(F)	yes	no	full
LNS+B(B)	no	yes	best
LNS+B(F)	no	yes	full
LNS+RB(B)	yes	yes	best
LNS+RB(F)	yes	yes	full

6.3 Experiments

6.3.1 Experiment 1: Large Neighbourhood Search

Research Question

What is the best variant of Large Neighbourhood Search?

Pre-experimental planning

All algorithms show similar performance on instances of the same size. Thus, we chose one instance from each size, skipping the smallest size that can be solved to optimality with BP in seconds. Therefore, we used 12 instances in this part of the evaluation; each result is the average of 5 runs.

Task

To select the LNS parameters, we thoroughly analysed the impact of different algorithmic components on a subset of instances from the benchmark set. We investigate the following components:

1. The repair mechanism: r_{BP} or r_{CG}

2. The initial destruction size k_0
3. Max. number of iterations without improvement $n_{\max}$
4. The destroyer selection
5. The role of adaptivity

Setup

For the experiments in this section, we used an setup with Java (OpenJDK 14.0.1) and CPLEX 12.10 to solve the master problem. The choice of solver does not impact the experimental results, as the behaviour of the MIP solver only differs for larger problems, while the subproblems in LNS are fairly small.

To have a metric quality that does not scale with the size of an instance, we evaluate the quality of a solution using the relative GAP compared to the best-known solution:

$$\mathrm{GAP}(S) = \frac{z(S) - z(S_{\mathrm{bks}})}{z(S_{\mathrm{bks}})} \cdot 100, \tag{6.8}$$

where S is the current solution and S_{bks} is the best-known solution among all methods evaluated in this chapter.

To investigate the impact of the initial destruction size, we tested sizes $k_0 \in \{5, 10, 15, 20\}$.

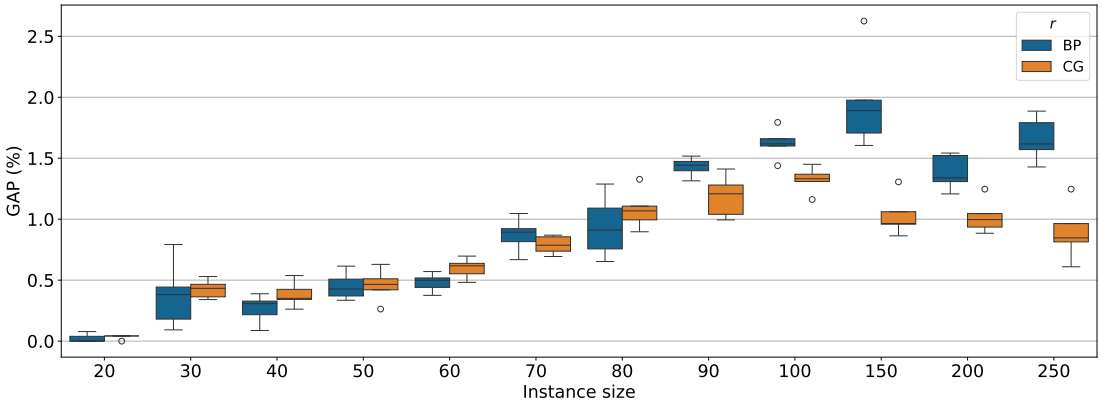
Next, we investigate increasing the size k every $n_{\max}$ iterations without improvement by 1, until reaching the upper bound $k_{\max} = 20$ or finding an improvement.

To investigate the impact of adaptivity, we conducted experiments by changing the parameter λ in Equation (6.5), considering all three destroy operators. We tested three different values for λ: $\frac{1}{3}$, $\frac{1}{2}$, and $\frac{2}{3}$.

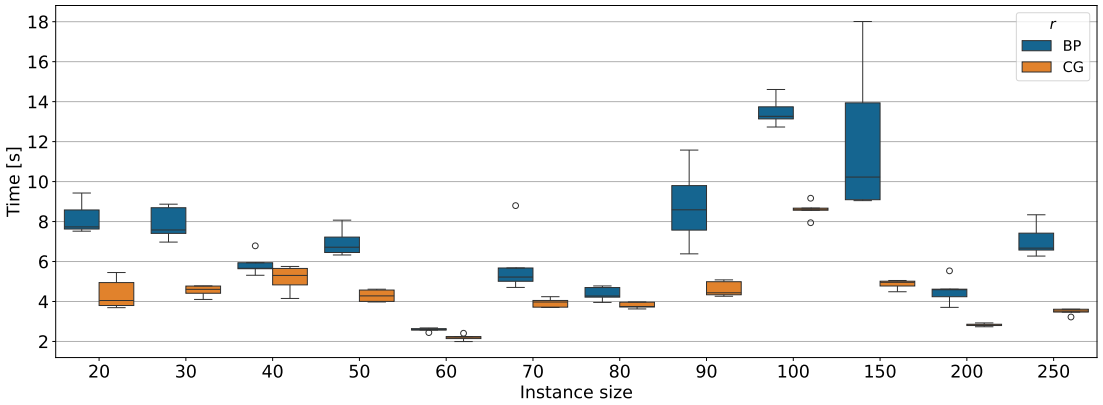
Observation

For fixed $k_0 = 10$, $n_{\max} = 50$, and equal selection of all destroyers, we compared r_{BP} and r_{CG}. Results are shown in Figure 6.1a. The performance of destroy and repair is rather independent from each other, which enables separate evaluation. Each repair operation has a maximum budget of 5 min, but is expected to usually terminate much faster. The performance is very similar for the smaller instances, but r_{CG} is clearly the better choice for larger instances, showing that the extra time used for repairing in r_{BP} is not justified. Figures 6.1b and 6.1c show that r_{BP} usually takes longer than r_{CG}. While the average time in both cases is under 10 s for most instances, r_{CG} shows significantly lower average values. r_{BP} often reaches the time budget of 5 min, while r_{CG} is mostly below 2 min, showing a much better worst case behaviour.

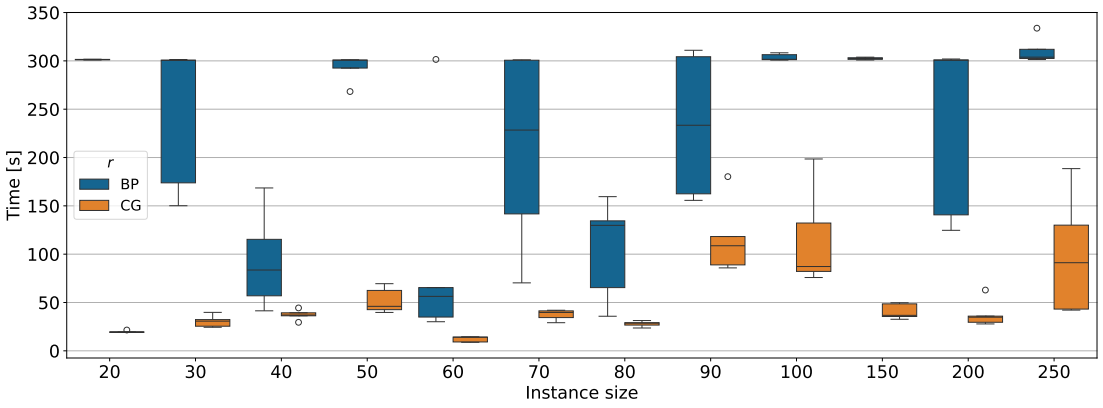
Regarding the initial destruction size, there are several options regarding the destroyers, first we investigated the initial destruction size k_0, fixing all other parameters. The size remained constant, r_{CG} and all destroyers are used, and $\rho_i = 1/3$ without adaptivity.



(a) GAP for different repair operators



(b) Average CPU-time of different repair operators



(c) Maximum CPU-time of different repair operators

Figure 6.1: Comparison of repair operators

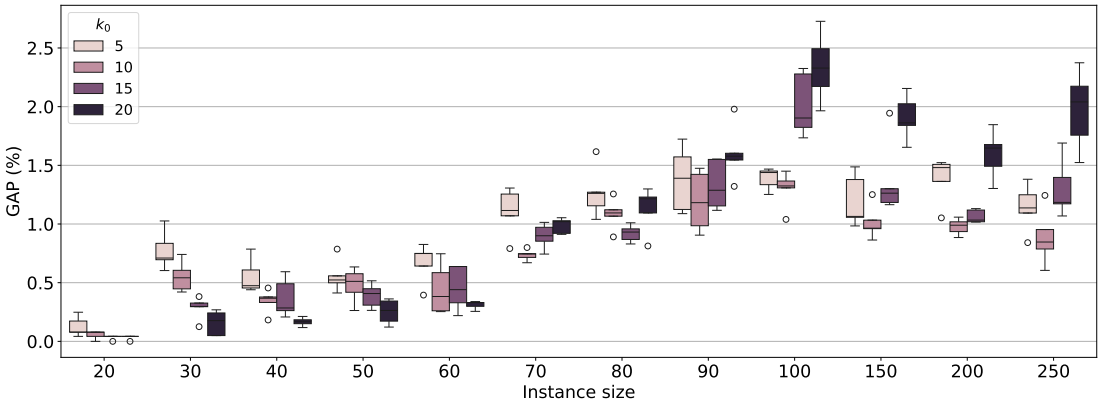


Figure 6.2: GAP for different values of k_0

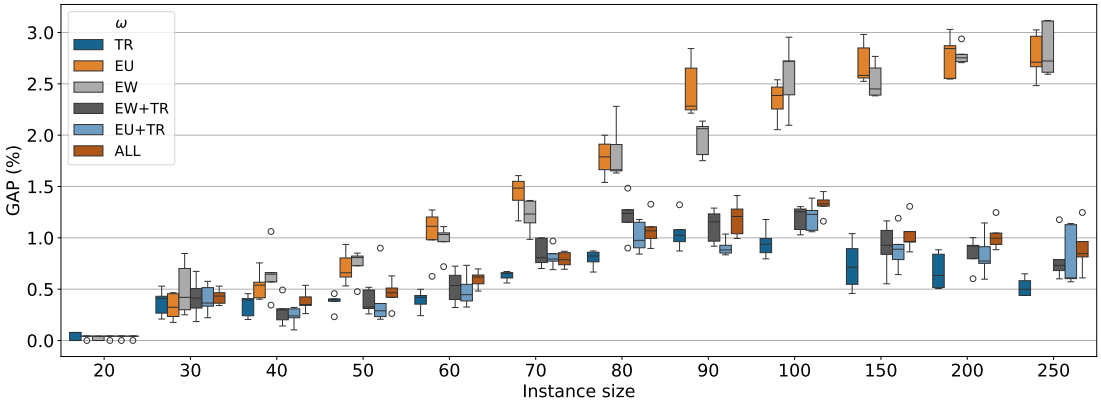


Figure 6.3: GAP for different subsets of destroyers

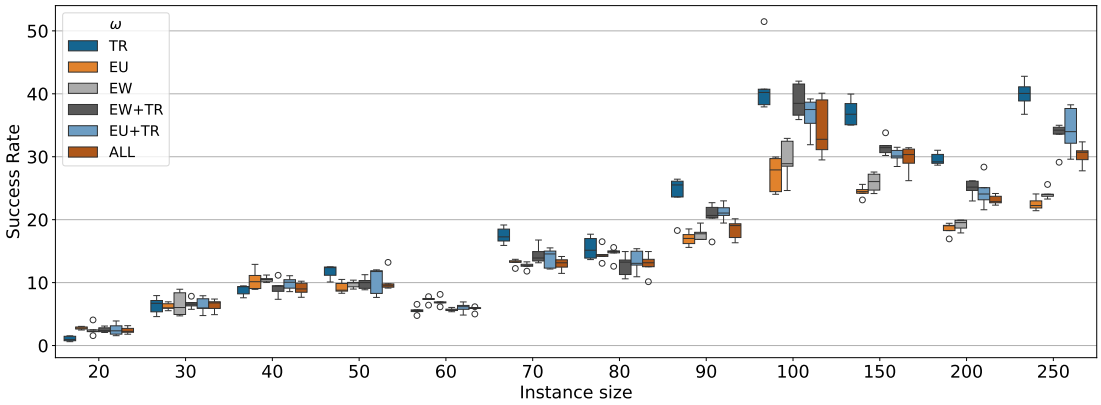
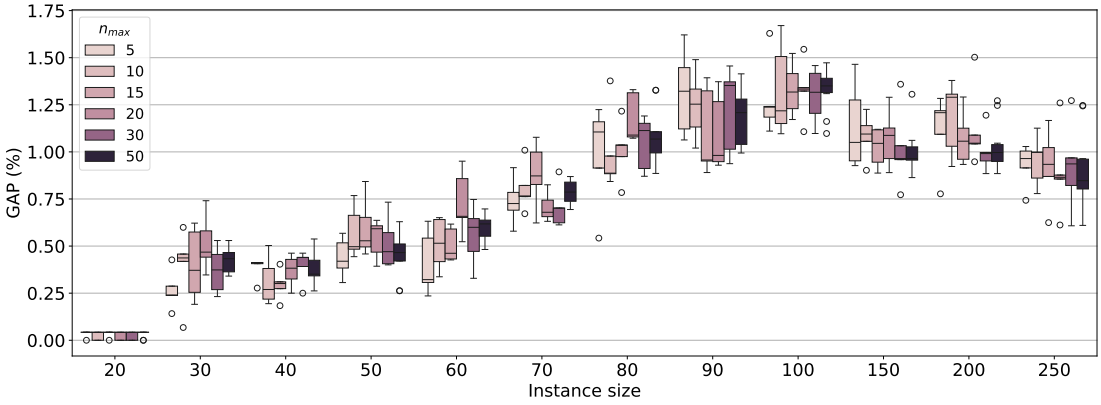
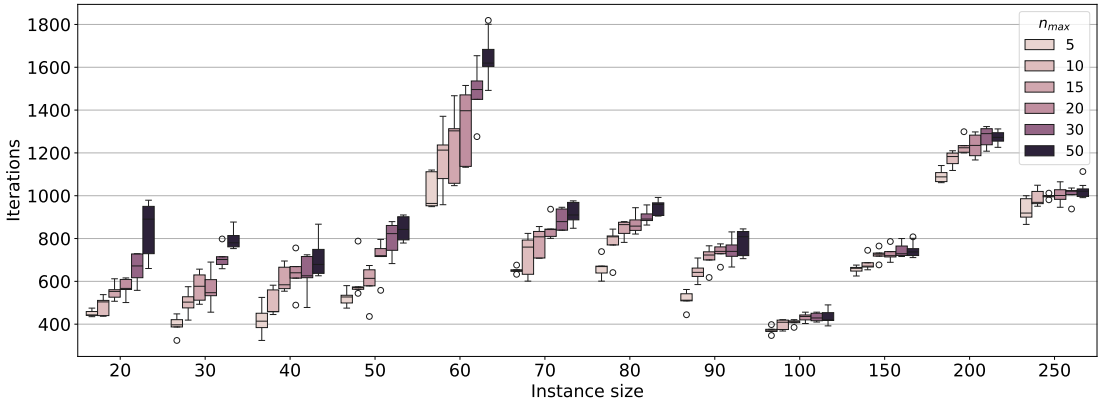


Figure 6.4: Success rate for different subsets of destroyers



(a) GAP for different values of $n_{\max}$



(b) Number of iterations for different values of $n_{\max}$

Figure 6.5: Comparison for different values of $n_{\max}$

Figure 6.2 shows the results. While $k_0 = 20$ performs slightly better for smaller instances, it is outperformed on larger instances. Overall, $k_0 = 10$ seems best for the large instances which are the main focus of LNS, therefore, we fix $k_0 = 10$.

Regarding the number of iterations without improvement, Figure 6.5a shows no significant difference among them. We decided to set $n_{\max} = 50$, since it still allows to increase the size when needed, but does not increase it very often. We tried starting with different values for k_0, but found similar results, the initial size is more important than the step.

Figure 6.5b shows the impact of $n_{\max}$ on the number of iterations (r_{CG} calls). Larger values of $n_{\max}$ imply less frequent size changes, so more iterations. Of note is that for the larger instances, the size barely changes, as improvements are frequently found even with the initial size until timeout.

Regarding the selection of the destroy operator, we tested all the 7 possible combinations

of them. Figure 6.3 shows that ω_{TR} has the biggest impact on the performance. It shows the best results on its own, with very similar results using it in any other combination, while all combinations without ω_{TR} show significantly worse performance, with higher divergence among larger instances.

The advantage of selecting employees that share tours is that the subproblems are more likely to allow meaningful optimisations. This hypothesis is further backed by the success rate (percentage of iterations where the current solution could be improved) in Figure 6.4, which shows that in general for larger instances more improvements in subproblems can be found, but especially using just ω_{TR} has a higher success rate than any other set of destroyers.

While only using ω_{TR} is the best choice, we further investigated several non-uniform weight distributions with high weights for ω_{TR}. Denoting the weights as $(\rho_{\omega_{\text{EU}}}, \rho_{\omega_{\text{EW}}}, \rho_{\omega_{\text{TR}}})$, we conducted experiments with $(5, 5, 90)$, $(10, 10, 80)$, and $(25, 25, 50)$. The first two where very similar to ω_{TR} only, while $(25, 25, 50)$ started to get slightly worse.

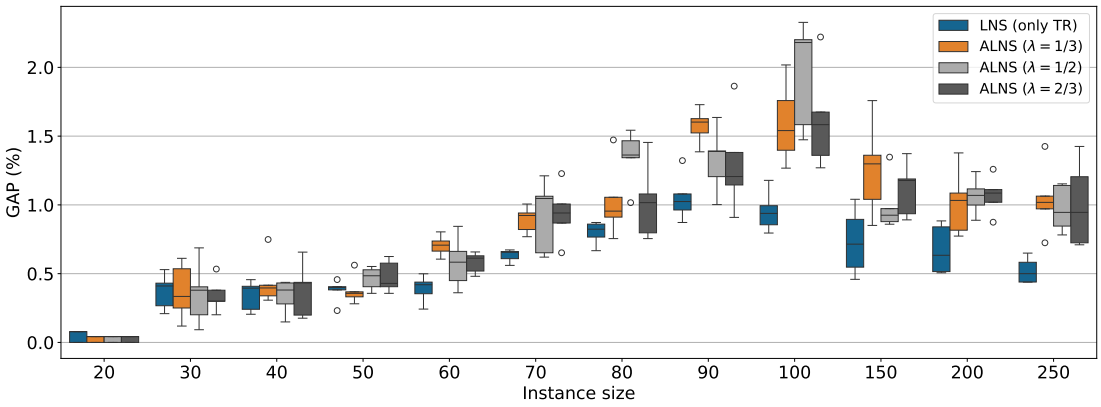


Figure 6.6: GAP for adaptive and static LNS

Figure 6.6 suggests that the adaptivity does not improve the average GAP with respect to the solely ω_{TR}, and different values of λ do not show significant difference.

Discussion

The best choice for the parameters is ω_{TR} as operator, an initial destruction size of $k_0 = 10$, and $n_{\max} = 50$ iterations without improvements.

6.3.2 Experiment 2: Integration of CG and LNS

This part is an integration part of the work of [KMVH21] together with Section 6.1.

Research Question

Can we achieve better results if we have a tighter integration between CG and LNS?

Task

More specifically, the questions we ask ourselves are

Q1: Do we benefit from storing and reusing the generated columns instead of throwing them away? For example, do we save time?

Q2: During the storing phase, is it worthwhile to store also non-optimal columns?

Q3: Does using an extra thread to solve the MIP problem in the background improve the overall performance?

Setup

We evaluate and compare seven distinct variants of the Large Neighbourhood Search algorithm with different levels of integration with Column Generation, summarized in Table 6.1.

The +R variants use the stored columns to initialize each subproblem according to Equation (6.7). The +B options apply a background MIP solver according to Section 6.2.2 on the set of stored columns $\hat{\mathcal{P}}$. For each combination, two different choices for the selection of columns to store (function `select`) are evaluated according to Section 6.2.3. The first option (B) is to store only the best columns S_i^* for each subproblem, the other option (F) to store the full set S_i.

To gain deeper insights into the strengths of the variants, we explore the following factors.

- **Number of Iterations:** We first examine the number of iterations performed by the LNS methods.
- **Repairing Time:** Next, we compare the CPU time required by the three methods. A key consideration is the trade-off between speed and resource consumption. Storing columns increases RAM usage but can speed up the evaluations. However, speed is not the only priority: slower evaluations that achieves greater improvements can lead to better solutions over time.
- **Success Rate:** We also measure the *success rate*, defined as the percentage of iterations in which the algorithm finds an improvement. For instance, a success rate of 50% indicates that the algorithm improves the solution in half of its iterations. While a high success rate is desirable, achieving larger improvements, even if less frequently, can still lead to superior solutions in the long run.
- **Convergence Plots:** We analyse the convergence behaviour of each variant by plotting its improvement over a fixed time budget (in our case, 1 h).

  Formally, a trajectory is represented as a sequence of pairs $(t, f(A, t, r))$, where $f(A, t, r, i)$ denotes the objective function value of the best solution found by

algorithm A within time t on its r-th run for instance i. For a fixed A and r, the sequence $\{f(A,t,r,i)\}_t$ is non-increasing, as the algorithm's best solution cannot deteriorate over time. We evaluate the empirical average over 10 runs:

$$\frac{1}{10}\sum_{r=1}^{10} f(A,t,r,i).$$

Note that, the average of two step functions is again a step function.

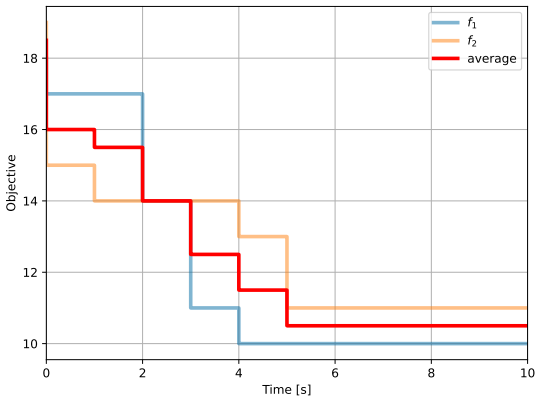


Figure 6.7: Two step functions and their average.

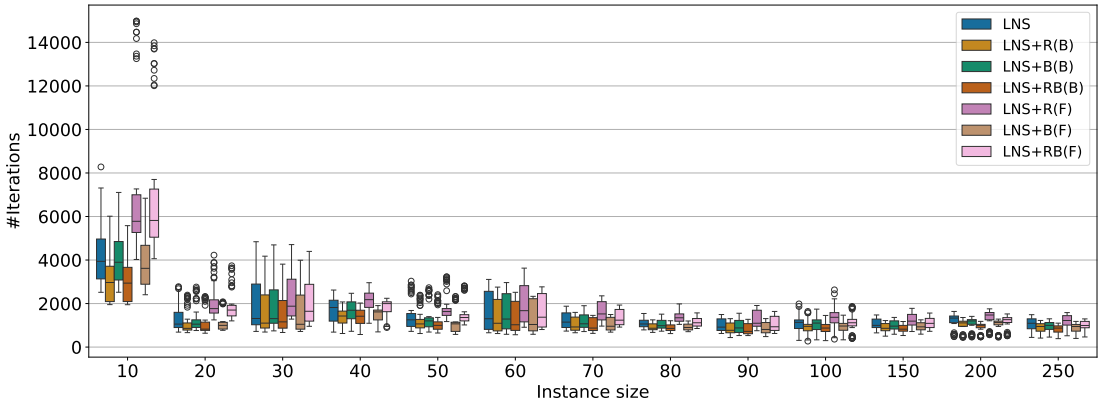
Observation

Figure 6.8 shows the number of iterations, average repairing time, and success rate for each of the methods, grouped by size. The general trend based on the size is clear. Smaller instances allow more iterations, since they are faster. For smaller instances, the success rate is low, indicating fast convergence, while for larger instances success rate is high, indicating less time is spent in local optima.

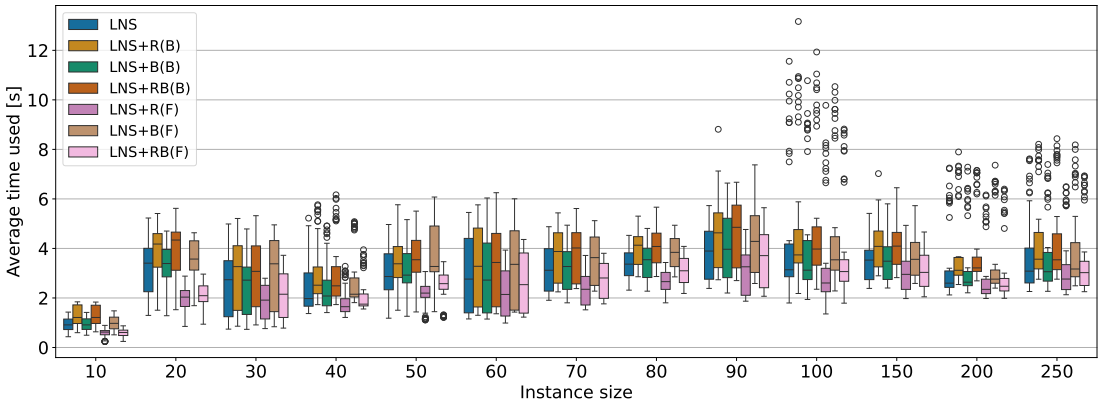
Between different variants of LNS, differences are small, but with several notable patterns. First, LNS and both +B variants show very similar results in all three metrics, which makes sense since the work in the background thread should not have significant impact on the main thread.

Next, both LNS+R(B) and LNS+RB(B) show fewer and slower iterations, while both LNS+R(F) and LNS+RB(F) show more and faster iterations. This is interesting since column reuse (+R) is supposed to speed up solving the subproblems. This only seems to work when adding the full set of columns (F), while only adding the best columns (B) actually seems to generate overhead instead of speeding up the search.

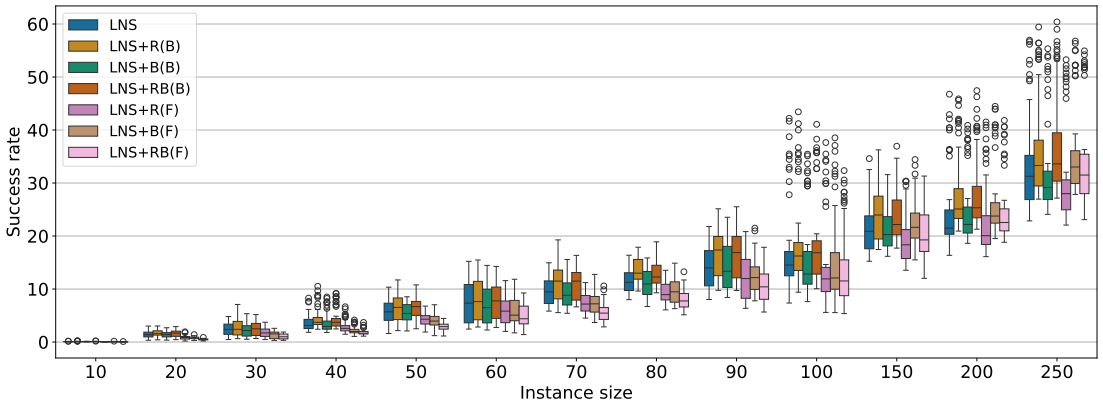
Figure 6.9 shows the GAP to the best known solution for the different variants. While the distinction regarding the metrics was small, the distinction regarding the GAP is very



(a) Number of iterations



(b) Average repairing time



(c) Success rate

Figure 6.8: Metrics of different LNS integrations

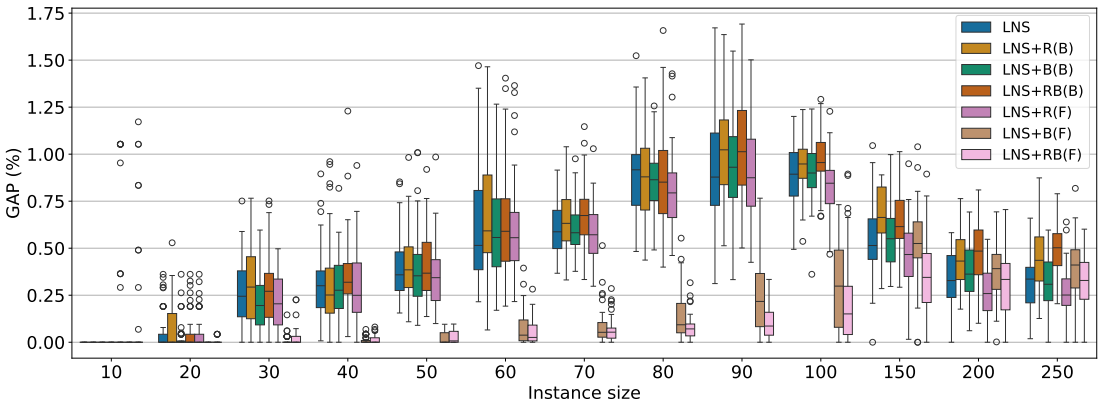
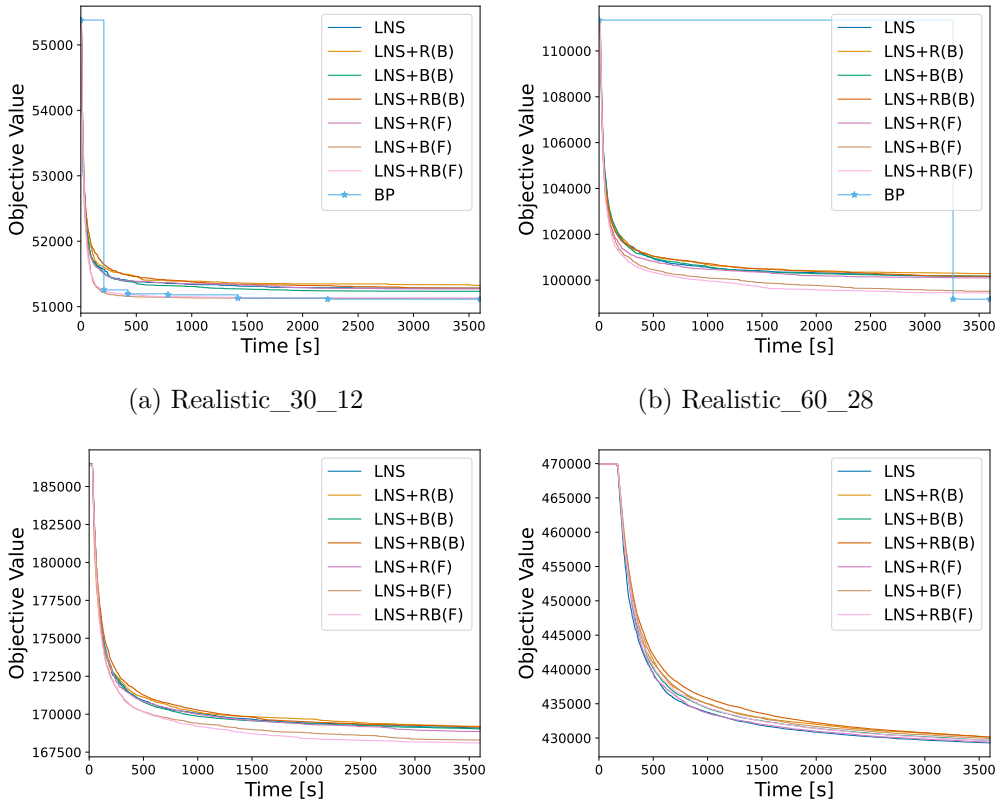


Figure 6.9: GAP for different LNS integrations



(a) Realistic_30_12

(b) Realistic_60_28

(c) Realistic_100_49

(d) Realistic_250_64

Figure 6.10: Convergence plots

clear. LNS+B(F) and LNS+RB(F) significantly outperform all other methods, which show similar performance among each other. This indicates that adding the background thread (+B) with the full set of columns (F) is the key combination to improve performance.

The shape of the graph in Figure 6.9 shows that this improvement is most beneficial for medium to large, but not very large instances. For small instances, all methods perform very well, with more notable distinctions starting at size 30. Separation grows larger up to around size 90, while for larger instances the GAP between methods shrinks, and methods perform very similar for sizes 200 and 250.

Of note is that for up to size 90, LNS+B(F) and LNS+RB(F) also show very low standard deviations, making them more stable than the other methods. LNS+B(F) starts to degrade around size 80, while LNS+RB(F) shows further benefits regarding deviations and also slightly better results than LNS+B(F) for sizes from 80 to 150.

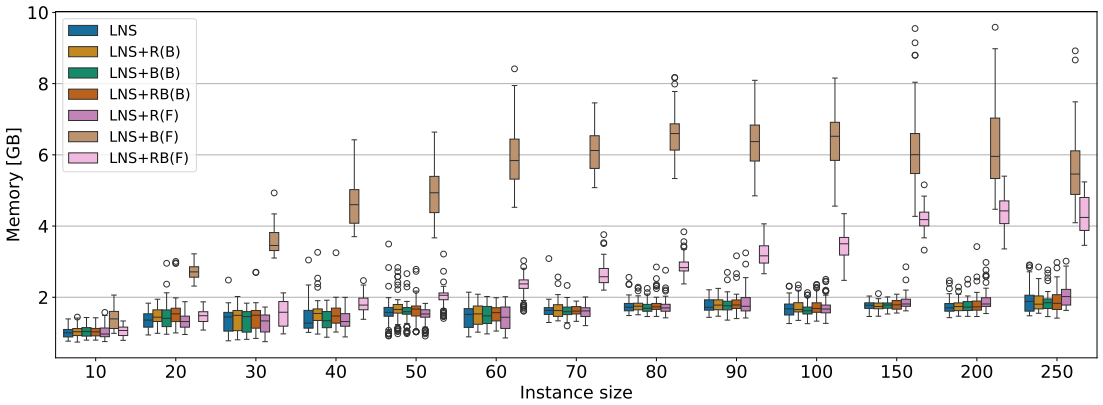


Figure 6.11: Memory usage of different LNS integrations

Figure 6.11 shows the memory use of the different LNS versions. The first conclusion is that all versions that do not combine the background thread (+B) with full storage (F) show very similar memory use. The pure storage, even of the full set of columns, is not significant, since the storage using bit sets in the implementation is very light-weight. However, there is significantly larger memory usage for LNS+B(F) and LNS+RB(F), especially for larger instances, showing that using the background thread with a larger number of columns is what needs extra memory.

While the quality of the results showed the large impact of the background thread and only minor improvements for column reuse, in comparison LNS+RB(F) uses significantly less memory than LNS+B(F). Upon closer investigation, the number of columns in the background thread grows much slower in LNS+RB(F) than in LNS+B(F). For example, in the instance realistic_100_49, the average number of columns differs by a factor of more than 3, from 2.0×10^6 in LNS+B(F) to 6.7×10^5 in LNS+RB(F). The suspected reason is that reusing the columns prevents the subproblems from generating a large number of suboptimal columns which would clutter the background thread. This is also consistent

with the slight performance benefit of LNS+B(F) regarding the GAP for sizes 80 to 150. While LNS+B(F) already has the background thread cluttered with too many suboptimal columns, LNS+RB(F) avoids this bottleneck for longer.

In order to analyse the behaviour of the different methods over their runtime, Figure 6.10 shows the convergence plots for some instances of different sizes to highlight different behaviours. The trajectories of the LNS versions are the average trajectories over 10 runs to avoid showing outlying behaviour. The first three instances all show the dominance of LNS+B(F) and LNS+RB(F) over the other LNS methods. The trajectories of these methods separate from the others very early in the search, and consistently stay ahead until the end. Figure 6.10a shows the behaviour of BP for smaller instances. While the results increase in steps (at the end of some nodes in the branching tree), the overall result and trajectory are comparable to the best LNS versions. Figure 6.10b shows the behaviour of BP for mid-sized instances. While it can outperform the LNS versions in the end, it does not provide a solution for a very long time, leading to a much better any-time behaviour of LNS. For the larger instances, BP only provides a solution with extra runtime, or using the background thread (BP+B). Figure 6.10d shows the behaviour of LNS for a very large instance. Here, the versions perform very similar, and the trajectories flatten out less, indicating that LNS is still improving regularly without hitting too many local optima, while the problem solved in the background thread gets larger and therefore harder to solve.

6.3.3 Experiment 3: Comparison of Different Methods

Research Question

How do our best LNS variants compare with state-of-the-art results from the literature?

Task

For our comparison, we consider BP, all our variants of LNS, as well as other results from the literature: CMSA [RKB+23], Simulated Annealing [KM20], Hill Climbing [KM20], and Tabu Search (Chapter 5). Since BP is deterministic, it is executed only once. The other three algorithms are run independently 10 times.

To add a well-founded view of the results, we perform a statistical analysis. The analysis is done in two steps: the omnibus test, and post-hoc test. In the omnibus test, we check whether at least one of the algorithms performs differently than the others. To do that, following the guidelines of Calvo [CS16], we use the Friedman test with Iman and Davemport extension. We formulate a Hypothesis H_0: *for every instance, the average objective function values are identical on all the algorithms.* With a significance level of $\alpha = 0.05$, the test rejects H_0 with a p-value smaller than 2.2×10^{-16}. Therefore, we can conclude that that there is strong statistical evidence that at least one algorithm performs differently than the rest. Thus, we conduct a post-hoc test to detect differences by pairs.

Setup

The setup is the same as in Section 6.3.

For statistical tests, we use the software tool scmamp [scm] ran with R (version 4.2.2).

Results/Visualisation

Table 6.2 shows minimum and average per size, with each row averaging the 5 instances of this particular size. The best performing method is highlighted in **bold**.

Table 6.2: Results for different solution methods grouped by size

	BP	LNS		LNS+RB(F)		CMSA	
Size		Min	Avg	Min	Avg	Min	Avg
10	**14 709.2**	14 709.2	**14 709.2**	14 709.2	14 728.2	14 867.4	14 879.7
20	30 299.4	30 294.6	30 312.3	30 294.6	**30 295.9**	30 695.8	30 745.9
30	**49 846.4**	49 867.0	49 966.4	49 841.0	49 851.2	50 731.4	50 817.2
40	67 017.0	67 023.8	67 165.7	66 957.6	**66 966.7**	68 394.8	68 499.9
50	84 338.8	84 415.4	84 583.7	84 251.4	**84 272.2**	86 219.0	86 389.2
60	99 754.6	100 006.4	100 246.4	99 673.6	**99 704.7**	102 596.2	102 822.9
70	118 337.6	118 512.2	118 698.9	117 991.2	**118 051.3**	120 935.6	121 141.9
80	134 925.8	134 827.6	135 178.8	133 983.4	**134 081.6**	138 406.8	138 760.3
90	150 292.8	150 282.6	150 721.6	149 340.2	**149 494.4**	154 692.6	155 078.3
100	168 554.2	166 397.4	166 811.5	165 331.4	**165 698.9**	171 159.4	171 786.7
150	273 629.2	254 742.2	255 449.1	254 195.8	**255 017.6**	263 079.2	263 387.7
200	380 518.0	337 479.0	338 226.9	337 221.8	**338 161.2**	348 608.6	349 017.0
250	520 063.4	426 908.4	427 866.7	426 739.4	**427 804.0**	438 811.4	439 234.5

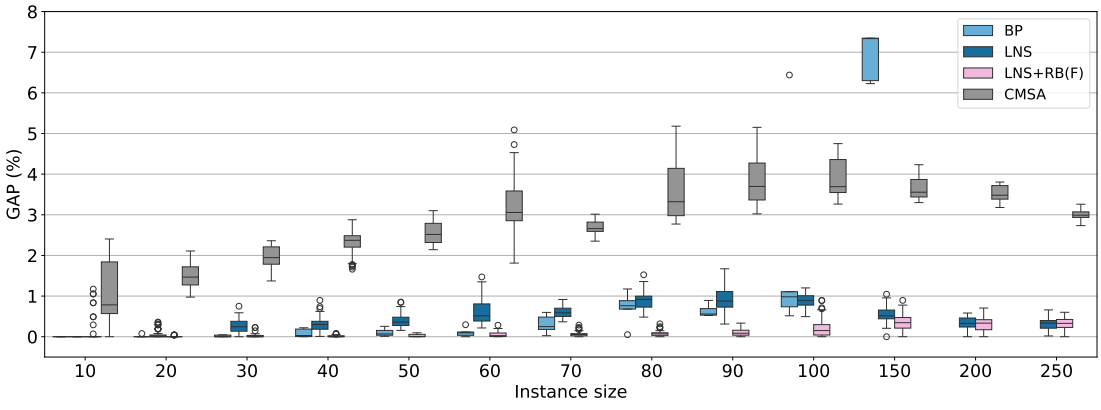


Figure 6.12: Results grouped by size.

We conduct an all pair-wise comparison with the Shaffer's static method. The results on all instances are graphically shown with a *Critical Difference* (CD) plot in Figure 6.13. Each considered algorithm is placed on the horizontal axis according to its average ranking for the instances (lower is better). The performances of those algorithm variants below the critical difference threshold (0.94) are considered statistically equivalent. In the CD plot, this is remarked by a horizontal bold bar that joins different algorithms.

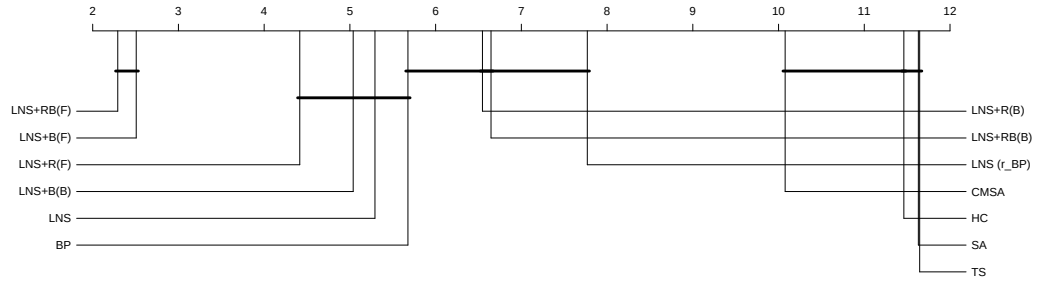


Figure 6.13: Critical Difference plot

Observation

Table 6.2 shows that CMSA outperforms all earlier metaheuristics on this problem, and was executed on a more powerful machine than ours (AMD Ryzen Threadripper PRO 3975WX processor with 32 cores, with a base clock frequency of 3.5 GHz, 64 GB of RAM). Figure 6.12 shows the gaps for these methods (average is used for stochastic methods).

According to Figure 6.13 there is no significant difference between LNS+RB(F) and LNS+B(F). They both outperform all other algorithms and LNS variants. All variants of LNS outperform the previous metaheuristic methods with significant difference.

Discussion

The results indicate that BP is the best method to use for small instances, as it can solve very small instances to optimality in a few seconds, and provides high-quality solutions with known low gaps to the optimum for up to size 30. BP considerably starts to struggle for large instances.

While results of BP still outperforms CMSA for instances of size up to 90, starting with size 40, the integrated method LNS+RB(F) already outperforms all other methods, and consistently provides the best results across all further sizes, making it the new state of the art for this problem.

Since LNS+RB(F) uses the background thread, and therefore more resources both in terms of computation time and memory, the best method regarding efficiency is LNS, which only needs one thread, and uses very moderate memory consumption. Still, in a very competitive environment, small improvements like those gained by LNS+RB(F)

translate to large savings over time, and the moderate additional resource consumption is worth the additional benefit in practice.

6.4 Conclusions

In this chapter, we presented a detailed study of Large Neighbourhood Search for the Bus Driver Scheduling Problem. We introduced a new destroy operator ω_{TR} that exploits the structure of the problem, and thoroughly evaluated several design choices for both the destroy and repair operators, as well as the use of adaptivity. The results scale very well even to very large instances, and can outperform previous metaheuristic solution methods. Finally, we proposed a new, tight integration between LNS and Column Generation (CG), where the best version of the integration LNS+RB(F) stores all columns from each subproblem in LNS, reuses them for future subproblems, and solves the integer master problem with all columns in a background thread. The evaluation showed that the background thread helps significantly improve the results of LNS, especially for mid-to-large-sized instances, while the column reuse is very beneficial in reducing the memory usage of the background thread. Overall, LNS+RB(F) is the new state of the art for instances of medium-to-large-sized size. While this chapter evaluates the methods on this complex version of BDSP, both the concepts introduced to solve high-dimensional Resource Constrained Shortest Path Problems (RCSPPs), and the integration of LNS and CG are general and can be applied to other scenarios, or to other optimisation problems.

While the results of this chapter show that LNS outperforms CMSA on the set of 65 instances, this is not always the case. In the next chapter, we will discover how to generate instances where the opposite is true and what are the weaknesses of our LNS hybrid technique.

CHAPTER 7

Generating New Instances and Analysing the Instance Space

About this chapter

This chapter describes and applies Instance Space Analysis (ISA) to the BDSP. The ISA is a methodology that allows us to gain more insight into the weaknesses and strengths of the algorithms. In addition, we get information about the quality and diversity of the benchmark instances.

This approach was originally published in a conference paper at PATAT2024 [MMKMSM24].

In Chapter 6 we showed that LNS seems to outperform CMSA; nevertheless, the reason was still unclear. To investigate the reason, we use ISA to show the weaknesses and strengths of the two solution methods. Instance Space Analysis [SMMn23] is a methodology that allows the diversity of a set of test instances to be visually examined, and insights into the strengths and weaknesses of algorithms, across the mathematically defined boundaries of the instance space, to be observed. First, we greatly extend an instance generator to be able to generate varied real-world-like and synthetic instances. This allows us to expand the previous set of instances from the literature.

We then present a set of features that describe the hardness of the instances. The features consider the structure of the instance, such as the average break length for each vehicle or the distribution of bus tours in the city. We observe that even if LNS outperforms CMSA in real-world-like instances, it does not for some synthetic ones.

Using ISA, each instance is projected into a 2D space based on selected features. We see clusters of instances in the instance space, and the real-world-like instances near the centre. The bus tour structure appears to have an impact on the performance of

the algorithms. Using this information, we can gain insights into the weaknesses and strengths of the two algorithms.

This chapter investigates how to (1) generate new *diverse* instances and (2) objectively assess how the state-of-the-art algorithms perform on those diverse instances.

In this setting, there is a set of 65 real-world-like instances (named *Realistic*) and performance results for two state-of-the-art algorithms: CMSA [RKB+23] and LNS [MKHM24]. Based on the 65 Realistic instances, LNS appears to outperform CMSA. However, we must scrutinise the diversity of these test instances and seek to ensure they span a range of instance characteristics before conclusions can be drawn about the merits of each algorithm.

7.1 Mathematical preliminaries

We state here some mathematical concepts required to follow the next sections. We start by defining the well-known **Gaussian** distribution.

Definition 7.1 (Normal distribution)**.** *The normal (or Gaussian) distribution with mean μ and standard deviation σ is denoted by $\mathcal{N}(\mu, \sigma)$, describing a bell-shaped probability centred in μ (the mean) and with a spread of σ^2 (the variance).*

Its probability density function is defined as

$$f_{\mathcal{N}}(x) = \frac{1}{\sqrt{2\pi}} \cdot \frac{1}{\sigma} \, \mathrm{e}^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{7.1}$$

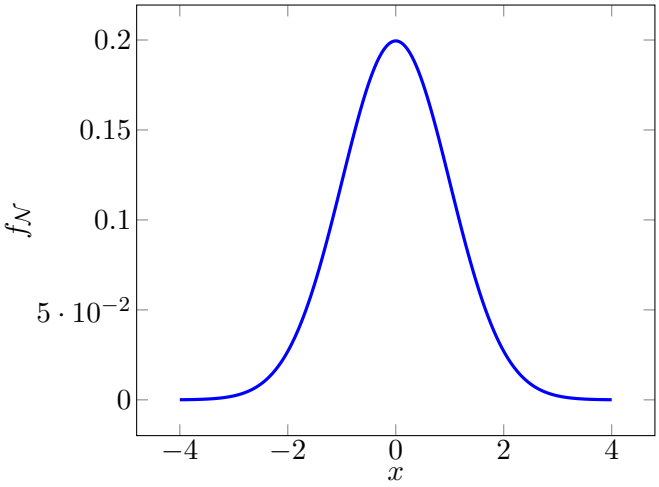
Definition 7.2 (Uniform distribution)**.** *Let $a, b \in \mathbb{R}$ be two real numbers. We denote the (continuous) uniform distribution on the interval $[a, b]$ by $\mathcal{U}_{[a,b]}$. Its probability density function is defined as*

$$f_{\mathcal{U}}(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in [a, b]; \\ 0 & \text{otherwise.} \end{cases} \tag{7.2}$$

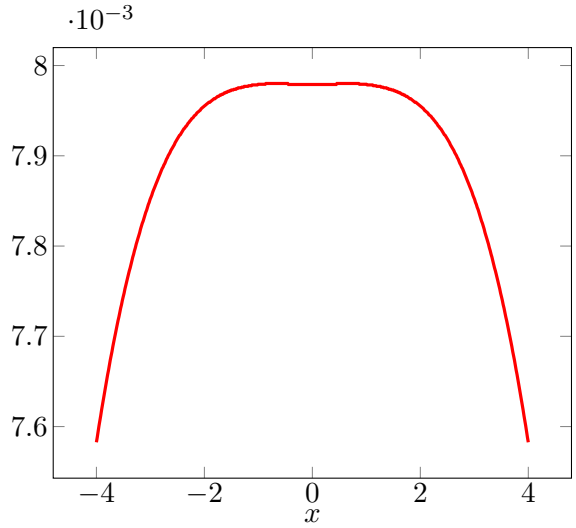
7.2 Instance Generator

Initially there were 65 publicly available real-world like instances for the BDSP.

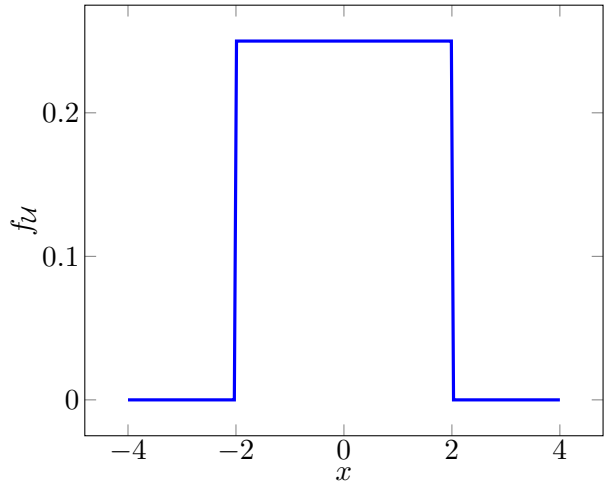
However, we can extend the generator to include different characteristics and, by this, to add diversity. In this section we aim to describe how we generate new instances. In order to generate an instance, we need the following steps:



(a) Normal distribution $\mathcal{N}(0, 1)$



(b) Normal distribution $\mathcal{N}(0, 50)$



(c) Uniform distribution $\mathcal{U}_{[-2,2]}$

Figure 7.1: Example of distributions

1. Generate a set of positions (bus stops).
2. Generate the distances between positions.
3. Generate extra work.
4. Create a demand distribution.
5. Generate bus tours that follow the demand distribution.
6. Divide each bus tour into several bus legs.

Positions

To create the bus stops and depot positions, we generate pairs $(x, y) \in \mathbb{R}^2$ in the plane. We generate the two coordinates following the normal distribution $\mathcal{N}(0, \sigma)$ for each axis.

For the instances `realistic_xx_y`, we generated the positions using standard deviation $\sigma = 50$. This means that for each axis, the Probability Density Function is given by:

$$f(x) = \frac{1}{50\sqrt{2\pi}} \cdot \mathrm{e}^{-\frac{x^2}{10\,000}} \tag{7.3}$$

We collect the two sets of generated positions (one for each axis) and create the set of positions $P = \{(x_i, y_i)\}_{i=1}^{n_{\text{stations}}}$, where n_{stations} is a parameter. The distance parameters are summarised in Table 7.1.

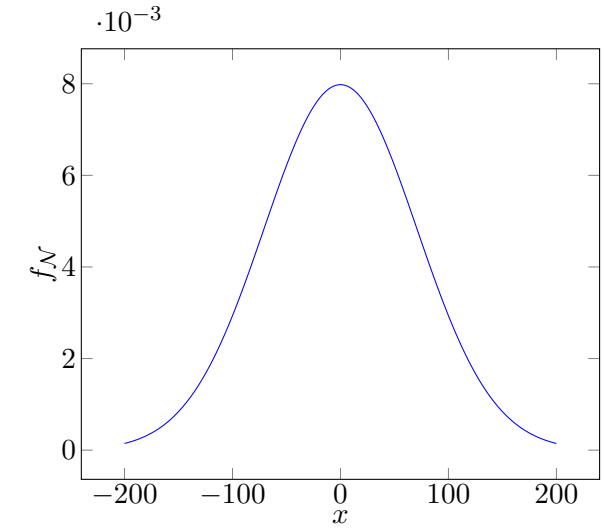
Table 7.1: Parameters used for the generation of positions. In the last column, the values used for `realistic` instances are stated.

parameter	name	value
σ	sigma	50
n_{stations}	sample size	10

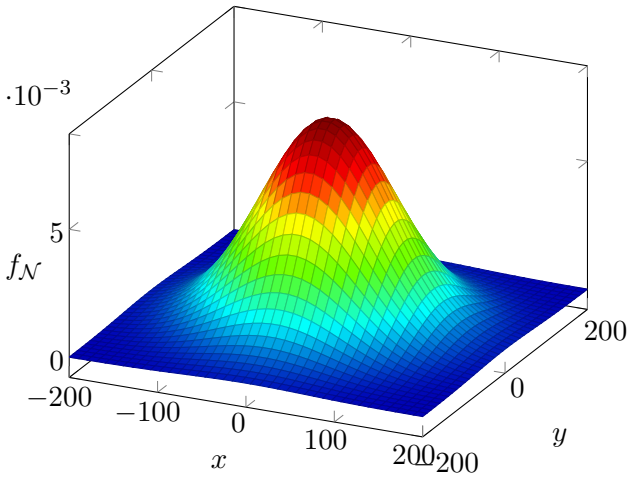
Extra work

As described in Chapter 3, for each position $p \in P$, there is an associated amount of working time for starting/ending a shift at position p. We represent these two parameters as $\texttt{startWork}_p$ and $\texttt{endWork}_p$

In the case of `realistic` instances, we set $\texttt{startWork}_0 = 15$ and $\texttt{endWork}_0 = 10$, assuming that the depot is at position $i = 0$. For all other positions $p \neq 0$, we have $\texttt{startWork}_p = \texttt{endWork}_p = 0$.

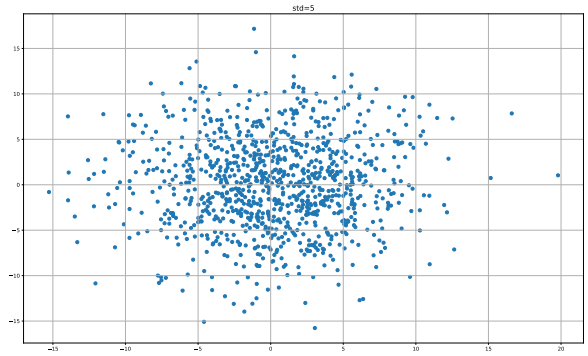


(a) Distribution on one axis

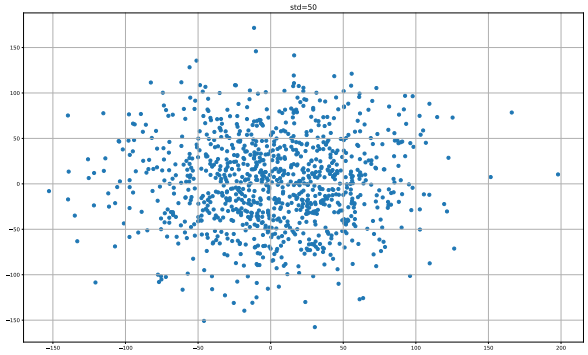


(b) Distribution in the 2-d plane.

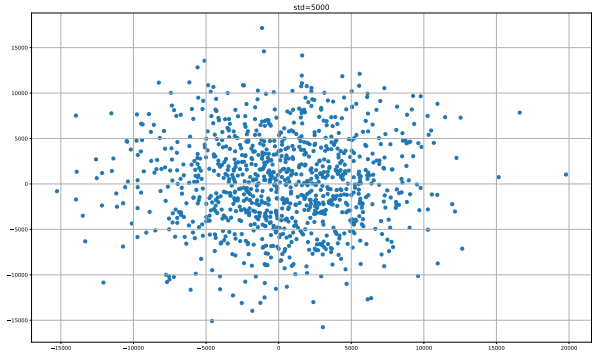
Figure 7.2: Normal distribution $\mathcal{N}(0, 50)$.



(a) $\sigma = 5$



(b) $\sigma = 50$



(c) $\sigma = 5000$

Figure 7.3: Samples of 1000 points drawn from the Normal distribution $\mathcal{N}(0, \mu)$, varying values of σ. The larger the standard deviation σ is, the more spread the distribution is.

Distances

We use a symmetrical time distance matrix $\boldsymbol{D} = (d_{ij})$ to model the time it takes for a driver to travel between two positions i and j [KM20]. This time takes into account not only the physical distance between i and j, but also factors such as traffic that can impact travel time.

For each $p \in P$, d_{pp} represents the time it takes to switch vehicles at the same position and is fixed to be 10 min[1].

When $i \neq j$, the distance d_{ij} is determined based on the probability p_{avail} of a direct connection between positions i and j. If the two positions have a direct connection, then we set their distance

$$d_{ij} = 10 + \lfloor q \cdot \text{dist}_{ij} \rfloor ,$$

where dist_{ij} is a term for the geometrical distance and $q \in \mathcal{U}_{[1,u_{\text{perturbation}}]}$ represents a uniform perturbation. If the two positions do not have a direct connection, then we set $d_{ij} = 1 \times 10^6$, since it is a number big enough to be used as an "infinite" distance.

Geometrical distance We introduce a new option to calculate the distance dist. For the instances `realistic`, the distance between two positions $i = (x_i, y_i)$ and $j = (x_j, y_j)$ was defined as

$$\text{dist}_{ij} := x_i^2 + y_j^2 \tag{7.4}$$

However, we also implement the Euclidean distance:

$$\text{dist}_{ij}^E := \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{7.5}$$

and the Manhattan one (L^1):

$$\text{dist}_{ij}^M = |x_i - x_j| + |y_i - y_j| \tag{7.6}$$

The parameters used in this calculation are summarised in Table 7.2.

$$d_{ij} = \begin{cases} 10 & \text{if } i = j; \\ 1 \times 10^6 & \text{if } i \neq j \text{ with probability } 1 - p_{\text{avail}} \\ 10 + \lfloor q \cdot \text{dist}_{ij} \rfloor & \text{if } i \neq j \text{ with probability } p_{\text{avail}}, \text{ where } q \in \mathcal{U}_{[1,\text{distvar}]} \end{cases} \tag{7.7}$$

The parameters used are listed in Table 7.2. In the last column, the values used for `realistic` instances are provided. Note that $p_{\text{avail}} = 0.9$ makes it extremely unlikely to have no direct connections between positions.

[1] Note that the term *distance* can be misleading. In fact, d_{ij} depends on the distance, but it should be rather interpreted as travel time.

Table 7.2: Parameters used for the generation of distances.

parameter	mean	value
p_{avail}	probability of direct	0.9
$u_{\text{perturbation}}$		1.2
M	large distance	99 999

Demand distribution

Our goal is to generate the demand distribution, which means creating a vector that specifies how many Bus Tours (vehicles) will be needed at different times of the day.

We can imagine to have a peak of bus needed during the morning time (for example, at 7:00). This means that we need to force many vehicles to be active in an interval that contains 7:00, for example [6:00, 8:00]. Moreover, we also need to know how many buses are necessary. We can think about this as a relative number (for example, if the average of needed buses is 1, we might need 1.5 buses at 7:00)

Let us define a peak as a moment of the day with very high demand. We formally model that, we need to introduce the concept of *peak*:

Definition 7.3 (peak)**.** *A* ***peak*** u *is defined as a* 5*-tuple:*

$$u = (u_{\text{time}}, u_{\text{height}}, u_{\text{after}}, u_\delta, u_\sigma) \in \mathbb{N}^4 \times \mathbb{R}, \tag{7.8}$$

where

- $u_{\text{time}} \in \mathbb{N}$ *is the time in which the peak reaches the maximum height*
- $u_{\text{height}} \in \mathbb{N}$ *represents the number of active buses needed at the peak*
- $u_{\text{after}} \in \mathbb{N}$ *represents the number of buses that need to stop and go back to the depot after the peak*
- $u_\delta \in \mathbb{N}$ *is the spread of the peak: no bus tours must start before* $u_{\text{time}} - u_\delta$ *and after* $u_{\text{time}} + u_\delta$
- $u_\sigma \in \mathbb{R}$ *is the standard deviation of the distribution of the peak (i.e., how large the peak is).*

For each peak, we have to know how many start times we have to generate (u_{height}) and how many end times, i.e., how many new buses must depart from the depot and how many should return to the depot right after the peak (u_{after}).

To do that, we can used the *relative* height, with respect to some base.

Let u be a peak and consider u_{height} and u_{after}. We can scale them with respect to some number k: $u_{\text{height}} = \lfloor k \cdot \alpha_u \rfloor$ and $u_{\text{after}} = \lfloor k \cdot \beta_u \rfloor$. where k is a number that we have to find.

For sake of clarity we remove the floor functions. The total number of tours n_{tours} that we generate is

$$n_{\text{tours}} \geq \sum_{u \in P} (u_{\text{height}} - u_{\text{after}}) = k \sum_{u \in P} (\alpha_u - \beta_u) . \tag{7.9}$$

This means that we can set

$$k = \frac{n_{\text{tours}}}{\sum_{u \in P} (\alpha_u - \beta_u) + \beta_{|P|}}. \tag{7.10}$$

For example, imagine we have a peak u with $u_{\text{height}} = 1.9$. This means that if the "base" if 1 bus needed, during the peak we need 1.9 buses.

How do we know what the base is? We know that we have n_{tours} possible vehicles available. Since we want to use all of them, we impose

$$n_{\text{tours}} = \sum_{u \text{ peak}} (\text{base} \cdot u_{\text{height}}) , \tag{7.11}$$

this implies that

$$\text{base} = \frac{n_{\text{tours}}}{\sum\limits_{u \text{ peak}} u_{\text{height}}}. \tag{7.12}$$

Bus tours

For us, a bus tour is a real interval $(s, t) \subset \mathbb{R}$, corresponding to one start and one end time. Later, we will divide each bus tour into several segments (bus legs).

To create the starting and ending times, we need a distribution of active vehicles (i.e., vehicles that are working during a certain time window). In our instances, we have three peaks of active vehicles that correspond to specific times of day.

These three peaks are represented by three specific times:

morning One early in the morning to get to the office/school/university. In our case `morningTime` = 420 (7:00)

afternoon One for lunch to get back home in case of part-time jobs/students. In our case `lunchTime` = 780 (13:00)

evening One later in the afternoon. In our case `eveningTime` = 1150 (19:10)

We can add two extra peaks (for example, one for the afternoon and another one for the night). We use the normal distribution for each peak, but we want to focus on the vehicles inside an interval $[\ell, u]$ that is why we first generate a set of points that follow the normal distribution. Then, if a point lies outside the interval $[\ell, u]$, we replace it with a random point inside the interval.

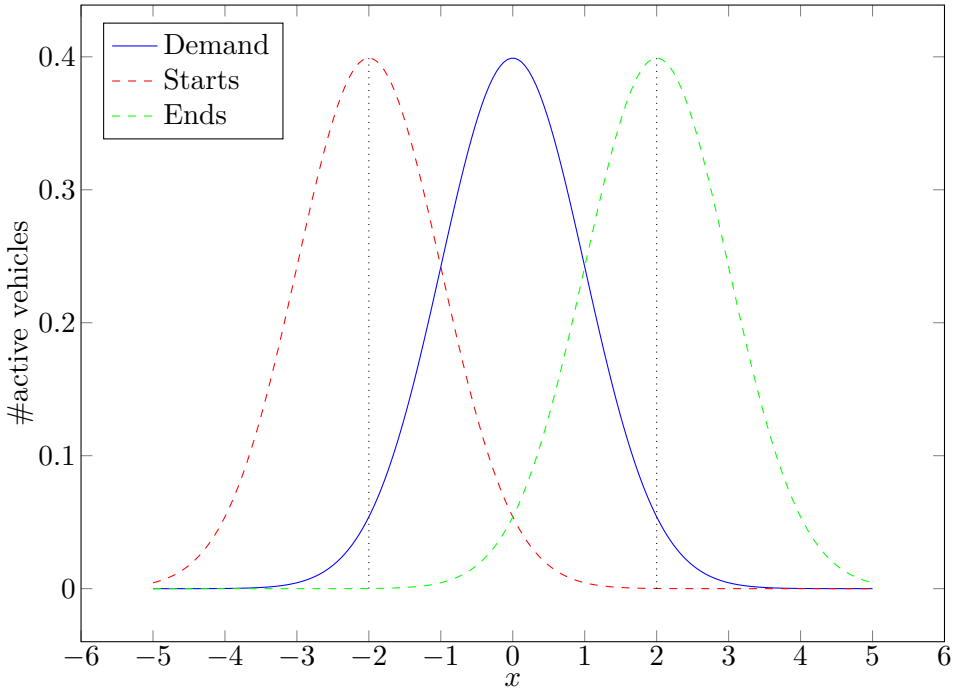


Figure 7.4: Distribution of the starting and ending time. The blue line represents the demand for vehicles over time, which follows a standard normal distribution $\mathcal{N}(0, 1)$. To ensure that there are enough vehicles active during the period $[-2, 2]$, we shift the distributions for starting times and ending times by a factor of 2, as indicated by the dashed red and green curves.

Table 7.3: Parameters for the demand distribution

parameter	name	value
`num_vehicles`	number of vehicles	10-250
`morningTime`	morning peak	07:00
`lunchTime`	lunch peak	13:00
`eveningTime`	evening peak	19:10

Bus Legs

It's important to note that the concept of bus tour does not necessarily correspond to a specific bus line. We are not considering different bus lines in our analysis. For example, bus line 22 may have multiple bus tours, each of which starts and ends at the same positions but at different times.

Assume to have a vehicle represented by the time interval $[s, t]$ (start and end times). We want to divide this interval into multiple pieces (bus legs).

Each bus leg ℓ is represented by a 5-tuple:

$$\ell = (\text{vehicle}_\ell, \text{start}_\ell, \text{end}_\ell, \text{startPos}_\ell, \text{endPos}_\ell),$$

The function generates a sequence of "legs" for a bus route, given a start and end time, and a vehicle index. Each leg corresponds to a section of the route between two stops. The function uses parameters such as leg duration, break duration, number of stops, and regularity of the schedule to generate the legs.

The function starts by reading various configuration parameters, such as the minimum and maximum leg duration, the maximum period of consecutive legs, the regularity of the schedule, and the number of stops. It then initializes various variables, such as the start time of the first leg, the duration of each leg and break, the number of stops per leg, and the previous station. It then enters a loop that generates legs and breaks for the bus route. Within the loop, the function randomly generates the duration of each leg and break, the number of stops for each leg, and the regularity of the schedule. It then generates a plan, which is a list of tuples, each containing the vehicle index, the start and end times of the leg, the previous station, and the current station. The loop continues until the end time is reached.

7.3 Instance Space Analysis

We introduced Instance Space Analysis (ISA) in Section 2.2 is a methodology proposed by Smith-Miles et al. in 2014 [SMBWL14] that extends the algorithm selection problem framework of Rice [Ric76]. The current form is described in the article from 2023 by Kate Smith-Miles and Mario Andrés Muñoz [SMMn23].

7.3.1 Problem Subset I

The original set of 50 instances from Kletzander and Musliu [KM20] was later [KMM22] extended with 15 new (again real-world-like) instances. The instances are publicly available[2]. For these instances, the number of bus tours ranges from 10 bus tours (about 70 legs) up to 250 (about 2300 bus legs). These instances are build to reproduce particular properties seen in a specific industrial use-case, however, in other settings

[2] https://cdlab-artis.dbai.tuwien.ac.at/papers/sa-bds/

involving different locations or rule sets, real-world instances might exhibit very different properties.

To cover a larger portion of the instance space, we greatly extended the original instance generator. It can generate instances with a larger number of bus stops and more diverse distributions of bus legs and tours during the day. The generator uses 44 parameters.

We changed some of this parameters (one at time) and we then generated 219 new diverse instances. The new instances are divided into 12 distinct classes. A brief description of the types is given in Table 7.4.

Table 7.4: The 12 instance classes. The third column gives the value for the existing benchmark instances.

Name	Characteristic	Standard
breakMax	No breaks between two consecutive bus legs	$[3, 35]$ min
distanceAvailability	The probability that 2 stops are connected is 0.1	0.9
distanceVariation	Add uniform $\mathcal{U}_{[1,100]}$ distance perturbation	$\mathcal{U}_{[1,1.2]}$
legRegularity	Probability of reusing the last leg is 0.1	0.9
numStations	There are 1000 bus stops	10
morningPeak	Morning peak is 5 times the regular demand	1.8
legPeriodMax	Max number of break lengths in use per tour is 5	3
shortLeg	Every leg length is in the interval $[5, 15]$ min	$[20, 60]$ min
gridSpread	Bus stops drawn using the distribution $\mathcal{N}^2(0, 1000)$	$\mathcal{N}^2(0, 50)$
legMax	The maximum leg length is 240 min	60 min
legMin	The minimum leg length is 5 min	20 min

Figure 7.5 shows the demand distribution of two instances. In both cases there is a significant morning peak when both employees and students need numerous buses within a brief period, followed by a decrease in activity. With the instance generator, we can create instances like in Figure 7.5b, where the peak in the morning is extremely high.

7.3.2 Algorithm Space $\mathcal{A}$

We ran Instance Space Analysis with two algorithms from the literature.

The first algorithm is a matheuristic algorithm: Construct, Merge, Solve and Adapt (CMSA) [RKB+23]. The second algorithm is the LNS described in Chapter 6.

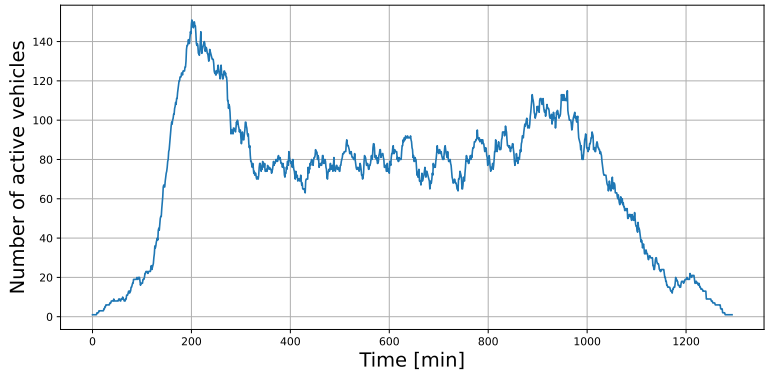
7.3.3 Feature Space $\mathcal{F}$

We collect a set of 84 features, described in Table 7.5.

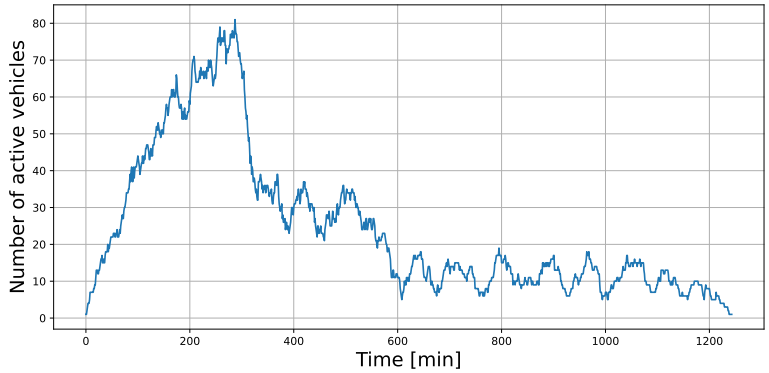
A special feature is the number of relative relief opportunities of an instance, defined as follow.

Table 7.5: Set of 84 features used in Meta-Data. With *glorious seven* we mean the seven descriptive statistics: Max, Min, Average, Median, Std, First quartile and Third quartile.

Feature Name	Description
Size-related: Dimension of the problem (4 features)	
Number of Tours	Number of distinct bus tours of the problem
Number of Legs	Number of distinct bus legs of the problem
Number of Positions	Number of bus stops (positions) used
Number of Active vehicles	Max number of active vehicles during the day
Geometry: (1 feature)	
Average distance	Average distance between bus stops
Bus Tours: Glorious seven across all tours for each feature (35 features)	
Total Time per tour	Total span time for each tour
Number of breaks per tour	Number of breaks between consecutive legs
Number of proper breaks per tour	Number of breaks of ≥ 15 min for each tour
Number of legs per tour	Number of bus legs for each tour
Number of large legs per tour	Number of legs with length ≥ 2 h for each tour
Distributions: Glorious seven across all legs for each feature (14 features)	
Drive	Bus legs lengths
Breaks statistics	Length of breaks between consecutive legs
RRO: Max, Min, Avg, Median, Std across all positions (25 features)	
Max RRO	Max Number of RROs over the time horizon
Min RRO	Min Number of RROs over the time horizon
Mean RRO	Mean Number of RROs over the time horizon
Median RRO	Median Number of RROs over the time horizon
Std RRO	Std Number of RROs over the time horizon
Bin Packing Problem [LSMC20]**:** k is the longest leg length (5 features)	
Huge	Proportion of legs that have length $\lvert\ell\rvert > k/2$
Large	Proportion of legs with $k/3 < \lvert\ell\rvert \leq k/2$
Medium	Proportion of legs with $k/4 < \lvert\ell\rvert \leq k/3$
Small	Proportion of legs with $k/10 < \lvert\ell\rvert \leq k/4$
Tiny	Proportion of legs with $\lvert\ell\rvert \leq k/10$



(a) `realistic_250_61`



(b) `morningPeak_250_3`

Figure 7.5: Demand Distribution of active vehicles.

Definition 7.4 (Relative Relief Opportunity (RRO))**.** *Let $p \in P$ be a position and $t \in \mathbb{R}$. We define a* ***relative relief opportunity*** *in p at time t (in minutes) as the proportion of bus legs that are starting from position p in the time window $[t, t + 60\,\text{min})$:*

$$RRO(p,t) = \frac{|\{\ell \in L \mid startPos_\ell = p \wedge start \in [t, t+60)\}|}{|\{\ell \in L \mid startPos_\ell = p\}|}$$

Note that for each position $p \in P$, we can evaluate $\max_t RRO(p,t)$ and, consequently, $\max_{p \in P} \max_t RRO(p,t)$ that tells us what the maximum relief opportunity across the positions throughout the day is.

RRO plays an important role, since it is correlated to the distribution of bus legs, bus tours and positions at the same time.

7.4 Experiments

Research Question

With this experiment, we mainly want to answer two questions:

1. What are the strengths and weaknesses of CMSA and LNS?
2. Can we find instances were LNS achieves worse results than CMSA?

Task

In Chapter 6, we discover that even the plain version of LNS achieves better results than the implementation of CMSA. We want to understand the reason why this happens and, if possible, to find instances there the opposite is true.

Setup

We perform the Instance Space Analysis using the Matlab toolkit MATILDA [SMMN20]. The settings for MATILDA are the default settings, therefore the number of final features is set to 10, as it is the default parameter. The only exception is the *performance threshold* set as 0.0. This means that for us an algorithm a_1 is better than a_2 on instance i if $y(a_1, i) < y(a_2, i)$ where $y(a, i)$ means the average of algorithm a on instance i of the objective function values over 10 runs.

In summary, we have 284 instances from two distinct sources, 84 features described in Table 7.5, and two algorithms: CMSA [RKB+23] and LNS [MKHM24].

Regarding the algorithm runs, we set 5 minutes as timeout for both algorithms. All executions were performed on a cluster with 11 nodes using Ubuntu 22.04.2 LTS. Each node has two Intel Xeon E5-2650 v4 (max 2.20 GHz, 12 physical cores, no hyperthreading). For each run, we set a memory limit of 4.267 GB and use one thread.

The implementation is in Python, executed with PyPy 7.3.11. Column Generation is implemented in Java, using OpenJDK 20, and CPLEX 22.11 for the master problem.

$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} 0.2636 & -0.7982 \\ -0.1663 & -0.6664 \\ 0.6380 & -0.5291 \\ -0.7371 & -1.4743 \\ -0.6819 & -0.5257 \\ -0.4333 & -0.8723 \\ -0.7684 & 0.3775 \\ -1.3987 & -0.2818 \\ -0.4252 & 0.4900 \\ -0.6950 & 0.0500 \end{bmatrix}^{\mathrm{T}} \cdot \begin{bmatrix} \text{driveMax} \\ \text{driveMean} \\ \text{driveMedian} \\ \text{drive3rdQuartile} \\ \text{maxTotalTimePerTour} \\ \text{minTotalTimePerTour} \\ \text{stdMaxRro} \\ \text{stdMeanRro} \\ \text{stdMedianRro} \\ \text{stdStdRro} \end{bmatrix} \tag{7.13}$$

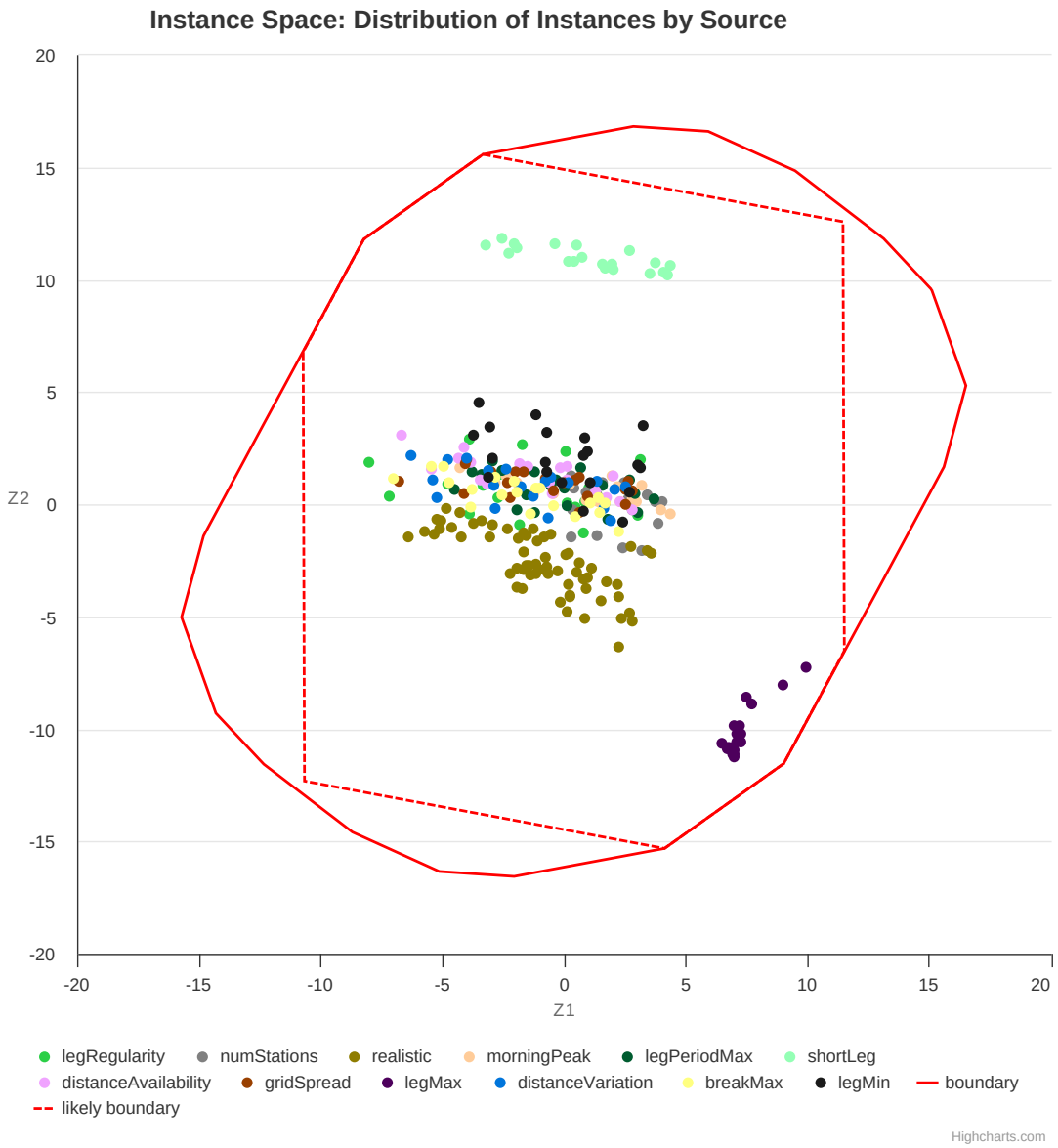
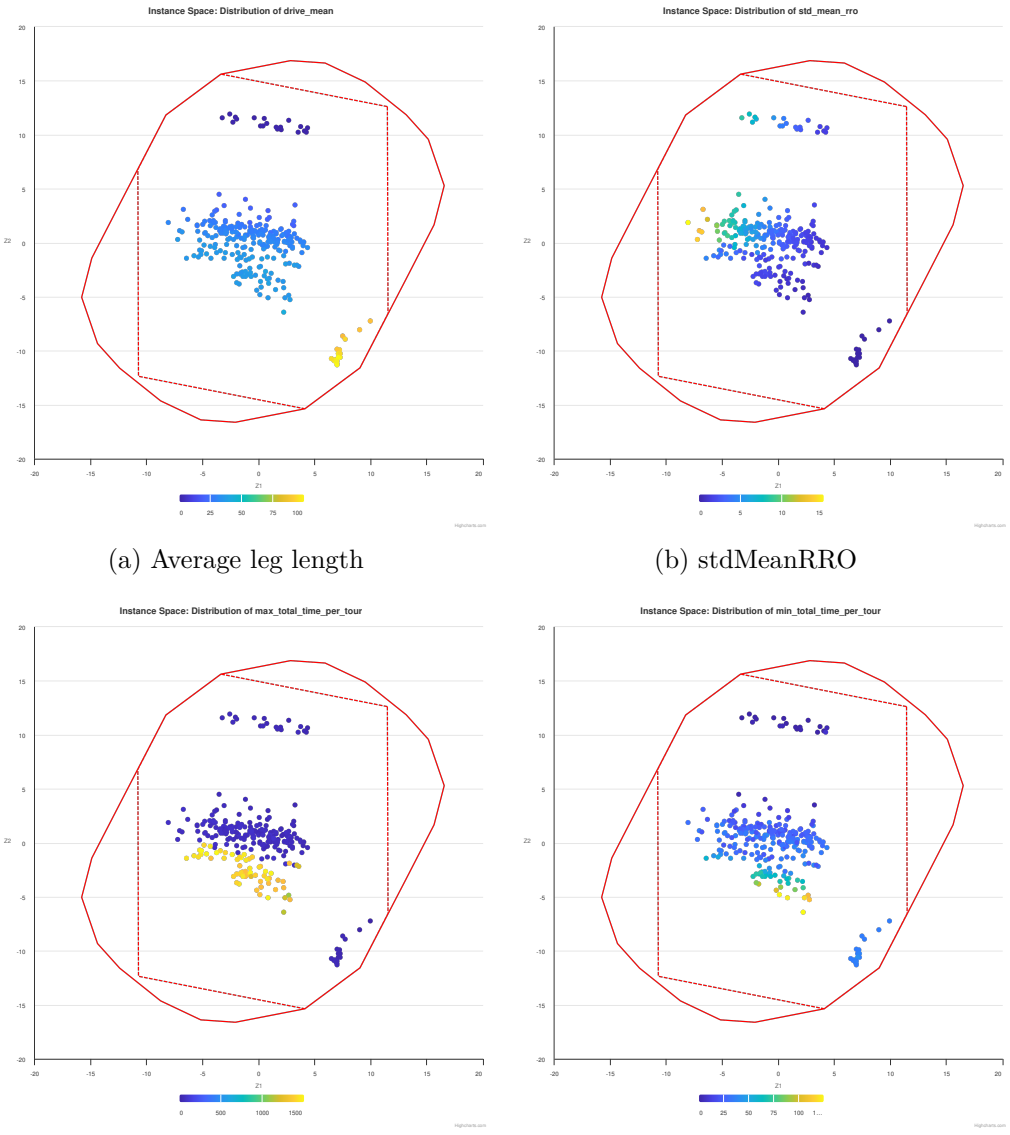


Figure 7.6: Bus Driver Scheduling Problem instance space defined by Equation (7.13). We recognise three main clusters: legMax, shortLeg and all the rest. We also observe that the Realistic instances (in brown) are in the middle of the instance space.



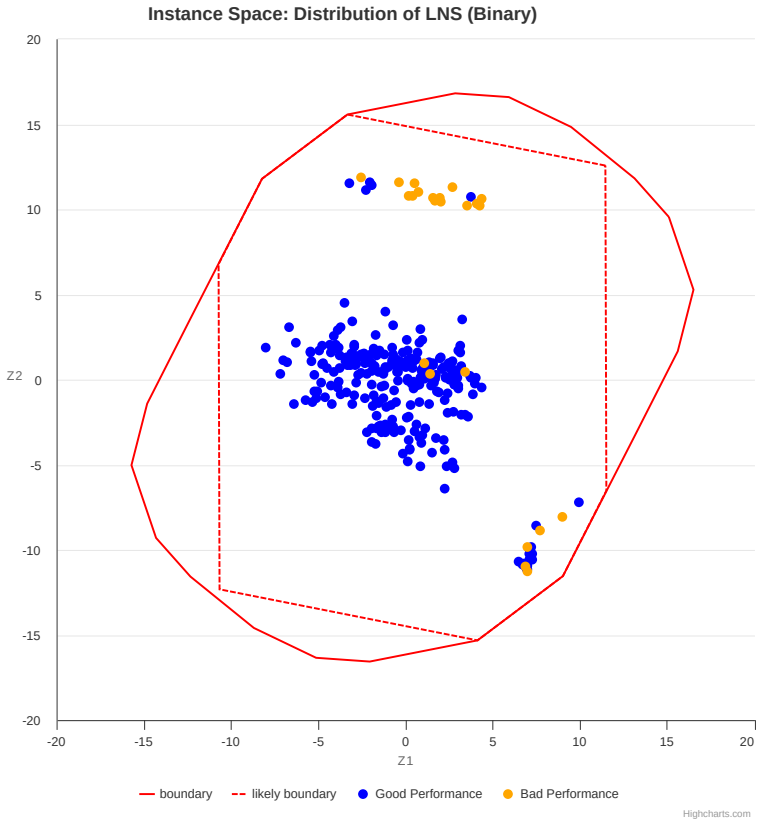
(a) Average leg length

(b) stdMeanRRO

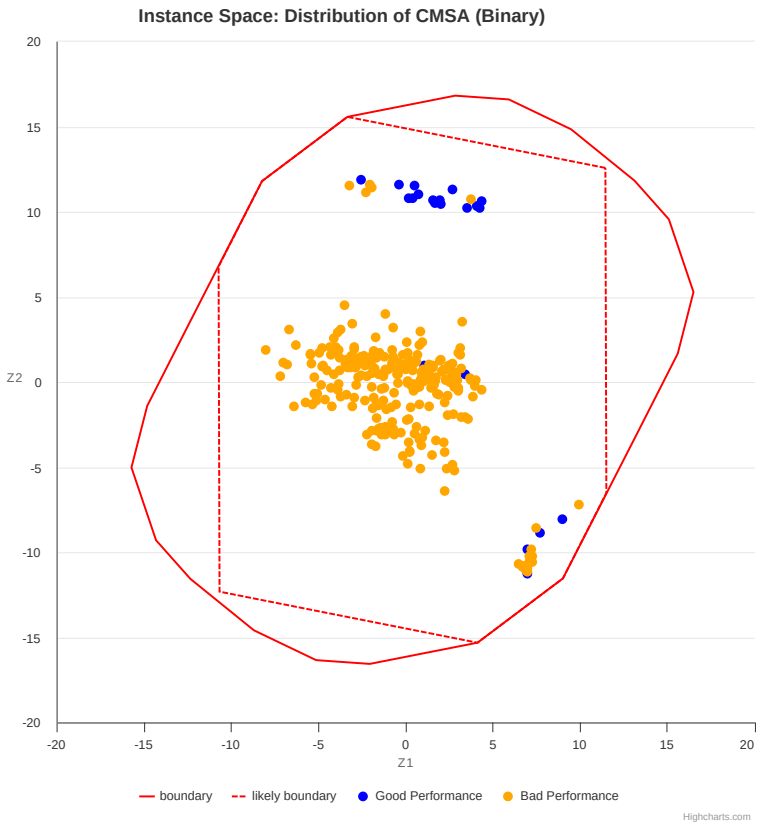
(c) Max Total Time x Tour

(d) Min Total Time x tour

Figure 7.7: Distribution of four of the selected features, from minimal (blue) to maximal (yellow) values of each scaled feature. Axes as defined by the Equation (7.13).



(a) LNS



(b) CMSA

Figure 7.8: Binary performance distribution. We see that CMSA performs better in some of the new generated instances close to the boundary.

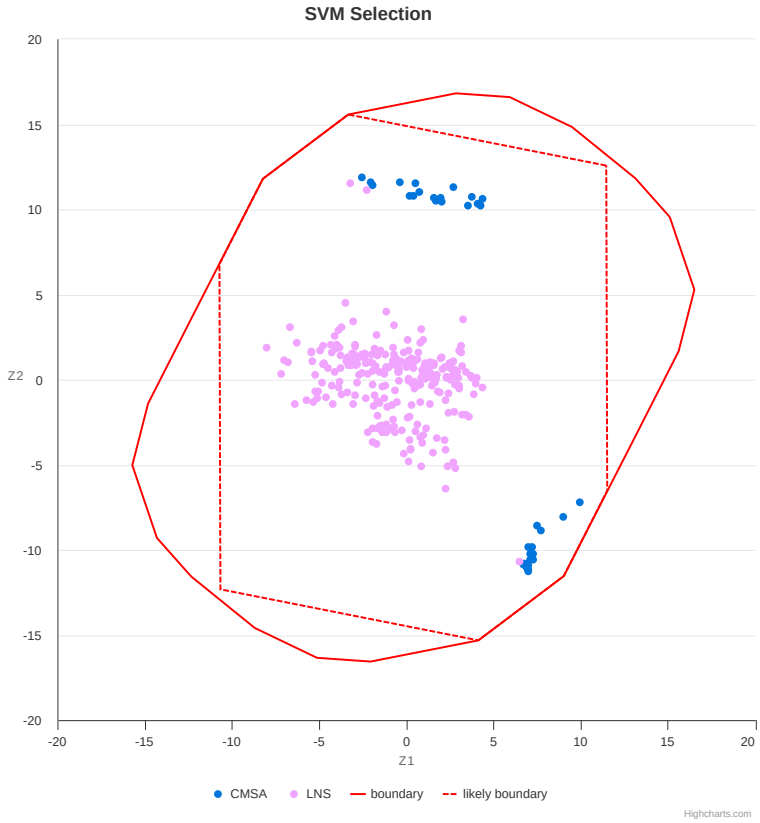


Figure 7.9: Recommended algorithms from SVM prediction models.

Equation (7.13) shows the projection matrix applied to the ten features after the preprocessing, that includes normalisation, and Box-Cox transformation [SMMn23]. We used ten features as it is the standard parameter value in MATILDA. We observe that four out of ten features describe the distribution of *drive*, that is, the leg length. Two features are related to the Total Time per Tour, i.e., the maximum and minimum total span of each tour, from the very beginning to the very end. The other four features describe the distribution of the Relative Relief Opportunities.

Figure 7.6 shows the distribution of the instance sources described in Table 7.4 across the instance space. We notice that Realistic instances are located around the centre of the instance space, meaning that the feature values are average. Moreover, shortLeg and legMax instances appear to be close to the theoretical boundary of the instance space. Those are instances where the leg length has drastically changed from the value of the Realistic instances. The red solid outer line represents the theoretical boundaries made by considering all the feasible combinations of features and their upper and lower bounds. The red dotted line instead is the likely boundary.

Figure 7.7 shows the distribution of four features out of the ten that are selected by

MATILDA. In Figure 7.7a we see that, as expected, the average length of bus legs is very high for legMax instances (where the leg lengths are drawn uniformly at random in the interval $[20, 240]$), whereas for shortLeg instances is extremely low (here the leg length are drawn uniformly at random in the interval $[5, 15]$). Figure 7.7b shows the distribution of the standard deviation (over the bus stops) of the minimum number of Relative Relief Opportunities during the day. This essentially is related to the number of possible changes/moves that we can do during the day for each bus stop. This value appears to be very low among the legMax instances, where the legs are usually large and, therefore the number of possible vehicle changes during the day is reduced. The other two images are about the total time per tour. In Figure 7.7c we see a clear distinction between Realistic instances and the new generated one. This is because the new generated instances have a lower maximum of length of bus tours. Figure 7.7d shows the Minimum total time per tour.

Figure 7.8 shows the binary performance distribution of the two algorithms. We observe that in the middle cluster, LNS performs better than CMSA. However, CMSA appears to have better results closer to the theoretical boundaries. Figure 7.9 shows the prediction of the Support Vector Machine, supporting this idea.

Discussion

We believe that the "structure" of the bus tour impacts the performance of the two algorithms. In particular, LNS removes all the bus legs in some selected bus tours. In contrast, CMSA randomly generates a number of greedy solutions at every iteration and, therefore, does not directly exploit the bus tours. LNS seems to perform better than CMSA for most of the instances (including Realistic), but not for all. We observe that CMSA gets better solutions for legMax and shortLeg instances. These instances have very short tours. Thus, LNS does not benefit much from removing all the legs associated with the same tour. Hence, CMSA (which does not explicitly depend on the structure of bus tours) provides good results with no significant difference from the others.

Thanks to ISA, we observe that there is still a considerably extended region between the four clusters in the instance space. This reveals opportunities to generate new instances to fill this gap. A first possible way to do that is by changing or adding parameters of the instance generator and trying to explore the instance space.

A more elaborated option is to fix a target (e.g., a portion of the instance space to fill up) and generate instances through a Genetic Algorithm that evolves new instances in the desirable region, as done by Smith-Miles [SMB15]. However, this procedure is problem-dependent and requires more investigation.

7.5 Conclusions

In this chapter, we have applied Instance Space Analysis to the Bus Driver Scheduling Problem for the first time. We evaluated the performance of two metaheuristic techniques

for the BDSP, providing insights into the strength of LNS and the boundaries of its good performance. We greatly increased the capabilities of the instance generator and extended the previous set of instances with new, diverse ones. We defined and evaluated a novel set of features, seeing which features help the most to explain algorithm performance.

Compared to the new instances, the realistic instances do not allow us to identify unique strengths and weaknesses of CMSA and LNSstudied (they are easy for both, and if one doesn't beat the other, it is very close). Consequently, their value as discriminating benchmarks, at least for these two highly competitive algorithms, is limited.

In the future, we want to fill the instance space by creating more instances that are even more diverse than the ones present now. At first, we will consider other public transportation systems, possibly located in different countries. Then, we will create instances with new combinations of parameters like long leg lengths and short bus tour lengths. Ideally, we want to use a Genetic Algorithm to automatically evolve the instances to fill up certain regions in the instance space. The goal is to perform automatic algorithm selection and outline the region of the instance space where one algorithm performs better than another. Thanks to this problem's structure, this will also be helpful for related problems such as vehicle routing.

Furthermore, we will test other solution methods using other quality metrics, such as the GAP from the best-known solution or the area under the curve of the trajectory of solutions found during the search.

CHAPTER 8

Conclusions

In this thesis, we studied the Bus Driver Scheduling Problem (BDSP), a combinatorial optimisation problem that arises in public transport planning and scheduling. We aimed to develop effective solution methods and a deeper understanding of the problem's structure rather than simply applying off-the-shelf algorithms from the literature.

8.1 Research contributions

We summarise below the main contributions of this work:

- **Metaheuristic methods based on Tabu Search** First, we developed metaheuristic methods based on Tabu Search local search. We thoroughly investigated the impact of those algorithms' parameters and components, and tuned the parameters to identifying which setting lead to better solutions.
- **Hybrid algorithm combining heuristics and Column Generation** Then, we designed a new hybrid method based on Large Neighbourhood Search (LNS) together with Column Generation (CG). This approach merges the flexibility of the heuristics with the speed and solution quality of CG, resulting in improvements over pure heuristic and pure CG.
- **Tight integration of Large Neighbourhood Search and Column Generation** We proposed a new, tight integration between Large Neighbourhood Search and Column Generation. In our approach, the solutions generated from each subproblem are stored to improve subsequent iterations, resulting in significantly faster convergence and better overall solutions.
- **Publicly available, diverse instance set** To gain insight into the strengths and weaknesses of existing algorithms, we generated a larger and more diverse set of test

instances. All of these instances have been made publicly available so that future researchers can run experiments and compare our results to their new approaches.

- **Instance Space Analysis for the BDSP** We proposed a set of instance features to characterise similarities and differences among test cases. These features include, for example, the number of bus tours, the average passenger demand, and network connectivity measures.

8.2 Results

From the extensive computational experiments and analyses presented in this work, we learnt the following lessons:

- **Good enough is better than optimal** Running experiments with LNS, we compared Branch and Price (B&P) and Column Generation (CG). We discovered that, even if CG does not achieve optimal results when solving the sub problem, it achieves solutions that are good enough and much faster than B&P. Even if the pure B&P remains the best method for smaller instances, our LNS results come very close on mid-sized instances while using fewer computational resources, making it the best choice for a wide range of realistic instance sizes. Note that the use of CG within LNS is very limited in literature, but despite the challenging subproblem very successful for this domain.

- **Using memory to save time** Even if our hybrid LNS+CG approach seems to work extremely well, the ultimate version was the tighter integration between these two elements. The evaluation shows that our approach provides new state-of-the-art results for instances of all sizes, including exact solutions for small instances, and low gaps to a known lower bound for mid-sized instances.

- **Generating new, diverse instances** When comparing our approach with the previous state-of-the-art method (CMSA), the experimental results indicate that our LNS achieves better results in the whole set of 65 instances in the literature. We decided to investigate further and we generated instances where the opposite is true.

- **Bus-tour–related features are frequently selected** We believe that the "structure" of bus tours affects our algorithms' performance. Specifically, LNS removes all legs belonging to some selected tours, whereas CMSA generates multiple greedy solutions at each iteration without explicitly exploiting this tour structure. As a result, LNS tends to outperform CMSA on most instances (including the Realistic set), but not universally. In particular, CMSA achieves better results on instances where tours are very short. We believe the reason is because LNS gains little by removing all legs of a compact tour.

8.3 Future directions

There are several options for continuing this work:

- First, the BDSP could be extended to consider uncertainties in the schedules or to deal with real-time data. At the same time, one could adapt our algorithms to work with other distinct sets of rules or collective agreements.
- Regarding Large Neighbourhood Search, a possible continuation considers the integration with CG. In particular, currently we have only two ways to select columns to store: either we select all the new columns generated or we select the best subset for each subproblem. However, one can explore better trade-offs, especially in the case of very large instances. We could also pass the best solution so far to the background solver in the second thread to use as the initial solution. Another future topic is applying the LNS+CG method to similar combinatorial problems, like vehicle routing.
- For Instance Space Analysis, we want to expand the instance space, filling the gap by generating many more, new, diverse instances. At first, we will consider other public transportation systems, possibly located in different countries. Then, we will create instances with new combinations of parameters. Ideally, we want to use a Genetic Algorithm to automatically evolve the instances to fill up certain regions in the Instance Space. The goal is to perform Automatic Algorithm Selection and outline the region of the instance space where one algorithm performs better than another. Furthermore, one can test other quality metrics, such as the GAP from the best-known solution or the area under the curve of the trajectory of solutions found during the search.

Overview of Generative AI Tools Used

The use of artificial intelligence (AI) tools in this thesis was conducted in adherence to established academic integrity protocols and ethical research standards. The used tools are:

- **Grammar checks**:
  - OpenAI's ChatGPT-4o (https://openai.com/chatgpt).
  - DeepSeek-R1-Lite-Preview (https://www.deepseek.com/).
  - Grammarly (https://app.grammarly.com/) for proofreading, stylistic refinement, and grammatical consistency.
- **Enhancing Literature Review**:
  - Perplexity AI (https://www.perplexity.ai/).
  - ConnectedPapers (https://www.connectedpapers.com/).
  - Google NotebookLM (https://notebooklm.google.com/).

No segment of this thesis was generated or substantively influenced by ChatGPT or other LLMs.

List of Figures

List of Tables

List of Algorithms

Glossary

Bus Leg A trip of a vehicle between two stops; in our implementation, a bus leg is a quintuple. The word *leg* denotes a portion of a journey, as specified by the definition 5 of Pearson Longman dictionary. Other common words are: *spell*, and *piece-of-work*. . 14

Bus Tour A sequence of bus legs assigned to one bus, starting with a departure from the depot and finishing with a return to the depot. Other common words are: *Running Board* (UK) and *Block* (North America).. 14, 78

CPLEX IBM® ILOG® CPLEX Commercial MIP Solver. https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer. 24, 57, 85

Exact algorithms Algorithms that, if they terminate, provide the optimal solution. For example, Branch and Price (B&P) or Branch and Bound. 25, 26

LSMC Large-Step Markov Chain, sometimes called Metropolis criterion. Assume we have the incumbent solution S. A new solution S' is accepted with probability $e^{\frac{z(S)-z(S')}{T}}$, where $T \in \mathbb{R}_+$ is a parameter that has to be tuned. 36

MATILDA Melbourne Algorithm Test Instance Library with Data Analytics. It is a software capable of performing Instance Space Analysis. https://matilda.unimelb.edu.au/. 85, 89, 90

OpenJDK Open Source implementation of Java. https://openjdk.org/. 57, 85

PyPy A fast and just-in-time implementation of Python. According to their website, PyPy is about 3 times faster than CPython 3.11 https://pypy.org/. 85

Acronyms

ALNS Adaptive Large Neighbourhood Search. 54

B&P Branch and Price. xi, 3–6, 24, 29, 30, 40, 42, 46, 47, 49, 51, 52, 55, 94, 105

BDSP Bus Driver Scheduling Problem. xi, 1–6, 13, 14, 16, 21, 24–26, 29, 46, 47, 49, 70, 71, 86, 90, 93–95, 100

CG Column Generation. xi, 3–6, 54, 55, 61, 62, 70, 85, 93–95

CMSA Construct, Merge, Solve & Adapt. 4–6, 26, 70–72, 82, 85, 88, 90, 91, 94, 100

HC Hill Climbing. 3, 40, 42, 46

ILP Integer Linear Programming. 26

ILS Iterated Local Search. xi, 3, 5, 6, 35–40, 42, 46, 47

ISA Instance Space Analysis. xi, 2, 4–6, 9, 71, 81, 82, 85, 90, 94, 95, 105

LNS Large Neighbourhood Search. xi, 3–6, 49, 50, 54–57, 60, 61, 63, 66, 67, 69–72, 82, 85, 90, 91, 93–95

RCSPP Resource Constrained Shortest Path Problem. 26, 70

SA Simulated Annealing. 3, 30, 36, 40, 42, 46

TS Tabu Search. xi, 3, 5, 6, 31, 34, 36, 38–40, 42, 46, 47, 67, 93

Bibliography

[BGM+10] Andreas Beer, Johannes Gärtner, Nysret Musliu, Werner Schafhauser, and Wolfgang Slany. An AI-Based Break-Scheduling System for Supervisory Personnel. *IEEE Intelligent Systems*, 25(2):60–73, March 2010.

[BJN+98] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

[BLR90] Jean-Yves Blais, Jacques Lamont, and Marc Rousseau. The hastus vehicle and manpower scheduling system at the société de transport de la communauté urbaine de montréal. *Interfaces*, 20(1):26–42, 1990.

[CdMdA+17] Ademir Aparecido Constantino, Candido FX de Mendonca, Silvio Alexandre de Araujo, Dario Landa-Silva, Rogério Calvi, and Allainclair Flausino dos Santos. Solving a large real-world bus driver scheduling problem with a multi-assignment based heuristic algorithm. *Journal of Universal Computer Science*, 23(5), 2017.

[Ced16] Avishai Ceder. *Public transit planning and operation: Modeling, practice and behavior*. CRC press, 2016.

[CMS19] Leandro do Carmo Martins and Gustavo Peixoto Silva. An adaptive large neighborhood search heuristic to solve the crew scheduling problem. In *Smart and Digital Cities*, pages 45–64. Springer, 2019.

[Com25] European Commission. Driving time and rest periods, 2025. Accessed: 2025-02-20.

[CS16] Borja Calvo and Guzmán Santafé. scmamp: Statistical Comparison of Multiple Algorithms in Multiple Problems. *The R Journal*, 8(1):248–256, 2016.

[CSSC13] Shijun Chen, Yindong Shen, Xuan Su, and Heming Chen. A Crew Scheduling with Chinese Meal Break Rules. *Journal of Transportation Systems Engineering and Information Technology*, 13(2):90–95, April 2013.

[DLFM11] Renato De Leone, Paola Festa, and Emilia Marchitto. Solving a bus driver scheduling problem with randomized multistart heuristics. *International Transactions in Operational Research*, 18(6):707–727, 2011.

[DS89] Martin Desrochers and François Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1):1–13, feb 1989.

[EGSNL23] Guillermo Esquivel-González, Antonio Sedeño-Noda, and Ginés León. The problem of assigning bus drivers to trips in a spanish public transport company. *Engineering Optimization*, 55(9):1597–1615, 2023.

[FMT87] Matteo Fischetti, Silvano Martello, and Paolo Toth. The fixed job schedule problem with spread-time constraints. *Operations Research*, 35(6):849–858, 1987.

[FMT89] Matteo Fischetti, Silvano Martello, and Paolo Toth. The fixed job schedule problem with working-time constraints. *Operations Research*, 37(3):395–403, 1989.

[FPW02] S Fores, L Proll, and A Wren. TRACS II: a hybrid IP/heuristic driver scheduling system for public transport. *Journal of the Operational Research Society*, 53(10):1093–1100, oct 2002.

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

[GP19] Michel Gendreau and Jean-Yves Potvin, editors. *Handbook of Metaheuristics*, volume 272 of *International Series in Operations Research & Management Science*. Springer International Publishing, Cham, 2019.

[ID05] Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer, 2005.

[IRDGM15] Omar J Ibarra-Rojas, Felipe Delgado, Ricardo Giesen, and Juan Carlos Muñoz. Planning, operation, and control of bus transport systems: A literature review. *Transportation Research Part B: Methodological*, 77:38–75, 2015.

[Kle22] Lucas Kletzander. *Automated solution methods for complex real-life personnel scheduling problems*. PhD thesis, Technische Universität Wien, 2022.

[KM20] Lucas Kletzander and Nysret Musliu. Solving large real-life bus driver scheduling problems with complex break constraints. *Proceedings of the International Conference on Automated Planning and Scheduling*, 30(1):421–429, June 2020.

[KMM22] Lucas Kletzander, Tommaso Mannelli Mazzoli, and Nysret Musliu. Metaheuristic algorithms for the bus driver scheduling problem with complex break constraints. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, July 2022.

[KMVH21] Lucas Kletzander, Nysret Musliu, and Pascal Van Hentenryck. Branch and price for bus driver scheduling with complex break constraints. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(13):11853–11861, May 2021.

[LEF+22] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

[LH16] Dung-Ying Lin and Ching-Lan Hsu. A column generation algorithm for the bus driver scheduling problem. *Journal of Advanced Transportation*, 50(8):1598–1615, 2016.

[LK03] Jingpeng Li and Raymond S.K. Kwan. A fuzzy genetic algorithm for driver scheduling. *European Journal of Operational Research*, 147(2):334–344, 2003. Fuzzy Sets in Scheduling and Planning.

[LMS19] Helena Ramalhinho Lourenço, Olivier C. Martin, and Thomas Stützle. *Iterated Local Search: Framework and Applications*, pages 129–168. Handbook of Metaheuristics, editors Michel Gendreau and Jean-Yves Potvin, Springer International Publishing, Cham, 2019.

[LPP01] Helena R. Lourenço, José P. Paixão, and Rita Portugal. Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation Science*, 35(3):331–343, 2001.

[LSMC20] Kelvin Liu, Kate Smith-Miles, and Alysson Costa. *Using Instance Space Analysis to Study the Bin Packing Problem*. PhD thesis, PhD thesis, 2020.

[MKHM24] Tommaso Mannelli Mazzoli, Lucas Kletzander, Pascal Van Hentenryck, and Nysret Musliu. Investigating large neighbourhood search for bus driver scheduling. In *34th International Conference on Automated Planning and Scheduling*, 2024.

[MMKMSM24] Tommaso Mannelli Mazzoli, Lucas Kletzander, Nysret Musliu, and Kate Smith-Miles. Instance space analysis for the bus driver scheduling problem. In *Proceedings of the Practice and Theory of Automated Timetabling*, pages 20–35, 2024.

[MT86] Silvano Martello and Paolo Toth. A heuristic approach to the bus driver scheduling problem. *European Journal of Operational Research*, 24(1):106–117, 1986. OR and Microcomputers Miscellaneous OR Applications.

[PLP08] Rita Portugal, Helena R. Lourenço, and José P. Paixão. Driver scheduling problem modelling. *Public Transport*, 1(2):103–120, nov 2008.

[PPÁME24] D Pardo-Peña, D Álvarez-Martínez, and J Escobar. A grasp algorithm for the bus crew scheduling problem. *International Journal of Industrial Engineering Computations*, 15(2):443–456, 2024.

[Pre15] Mike Preuss. *Experimentation in Evolutionary Computation*, pages 27–54. Springer, 2015.

[PS98] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[Ric76] John R. Rice. The algorithm selection problem. pages 65–118, 1976.

[RKB+23] Roberto Maria Rosati, Lucas Kletzander, Christian Blum, Nysret Musliu, and Andrea Schaerf. Construct, merge, solve and adapt applied to a bus driver scheduling problem with complex break constraints. In *AIxIA 2022 – Advances in Artificial Intelligence*, pages 254–267. Springer International Publishing, 2023.

[RMVP13] Ana Respício, Margarida Moz, and Margarida Vaz Pato. Enhanced genetic algorithms for a bi-objective bus driver rostering problem: a computational study. *International Transactions in Operational Research*, 20(4):443–470, 2013.

[RP06] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, nov 2006.

[scm] https://github.com/b0rxa/scmamp/tree/e435f9d48078f93ab49b23a19fdb6ef6e12ea5f9. Accessed: 2023-08=15.

[Sha98] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming — CP98*, pages 417–431. Springer Berlin Heidelberg, 1998.

[SK01a] Yindong Shen and Raymond S. K. Kwan. Tabu Search for Driver Scheduling. In G. Fandel, W. Trockel, C. D. Aliprantis, Dan Kovenock, Stefan Voß, and Joachim R. Daduna, editors, *Computer-Aided Scheduling of Public Transport*, volume 505, pages 121–135. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. Series Title: Lecture Notes in Economics and Mathematical Systems.

[SK01b] Yindong Shen and Raymond S. K. Kwan. Tabu Search for Driver Scheduling. In G. Fandel, W. Trockel, C. D. Aliprantis, Dan Kovenock, Stefan Voß, and Joachim R. Daduna, editors, *Computer-Aided Scheduling of Public Transport*, volume 505, pages 121–135. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[SMB15] Kate Smith-Miles and Simon Bowly. Generating new test instances by evolving in instance space. *Computers & Operations Research*, 63:102–113, 2015.

[SMBWL14] Kate Smith-Miles, Davaatseren Baatar, Brendan Wreford, and Rhyd Lewis. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12–24, 2014.

[Smi88] Barbara M Smith. Impacs-a bus crew scheduling system using integer programming. *Mathematical Programming*, 42(1):181–187, 1988.

[SMMN20] K. Smith-Miles, M.A. Muñoz, and Neelofar. Melbourne algorithm test instance library with data analytics (matilda), 2020. Available online.

[SMMn23] Kate Smith-Miles and Mario Andrés Muñoz. Instance space analysis for algorithm testing: Methodology and software tools. *ACM Comput. Surv.*, 55(12), mar 2023.

[SS15] Tiago Alves Silva and Gustavo Peixoto Silva. O uso da metaheurística guided local search para resolver o problema de escala de motoristas de ônibus urbano. *TRANSPORTES*, 23(2):105, August 2015.

[SW88] Barbara M. Smith and Anthony Wren. A bus crew scheduling system using a set covering formulation. *Transportation Research Part A: General*, 22(2):97–108, 1988.

[TK13] Attila Tóth and Miklós Krész. An efficient solution approach for real-world driver scheduling problems in urban bus transportation. *Central European Journal of Operations Research*, 21(S1):75–94, June 2013.

[WCM24] Mengtong Wang, Shukai Chen, and Qiang Meng. Robust safety driver scheduling for autonomous buses. *Transportation research part B: methodological*, 184:102965, 2024.

[WFK+03] Anthony Wren, Sarah Fores, Ann Kwan, Raymond Kwan, Margaret Parker, and Les Proll. A flexible system for scheduling drivers. *Journal of Scheduling*, 6:437–455, 2003.

[Wir19] Wirtschaftskammer Österreich. Kollektivvertrag für private autobusbetriebe 2019. https://www.wko.at/kollektivvertrag/kv-private-autobusbetriebe-2019, 2019. Accessed: 2023-10-02.

[WM14] Magdalena Widl and Nysret Musliu. The break scheduling problem: complexity results and practical algorithms. *Memetic Computing*, 6(2):97–112, June 2014.

[WR95] Anthony Wren and Jean-Marc Rousseau. Bus driver scheduling—an overview. In *Computer-Aided Transit Scheduling: Proceedings of the Sixth International Workshop on Computer-Aided Scheduling of Public Transport*, pages 173–187. Springer, 1995.

[Wre04] Anthony Wren. Scheduling vehicles and their drivers-forty years' experience. Technical report, University of Leed, 2004.