TU WIEN Informatics

Advancing State Space Search for Static and Dynamic Optimization by Parallelization and Learning

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Nikolaus Frohner, BSc
Matrikelnummer 00526087

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl

Diese Dissertation haben begutachtet:

Christian Blum	Sophie Parragh

Wien, 30. März 2023

Nikolaus Frohner

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

TU WIEN Informatics

Advancing State Space Search for Static and Dynamic Optimization by Parallelization and Learning

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Nikolaus Frohner, BSc
Registration Number 00526087

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl

The dissertation has been reviewed by:

Christian Blum	Sophie Parragh

Vienna, 30th March, 2023

Nikolaus Frohner

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Nikolaus Frohner, BSc

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

Wien, 30. März 2023

Nikolaus Frohner

Danksagung

Zuallererst möchte ich mich bei meinem Betreuer Günther Raidl für die tolle Zusammenarbeit über all die Jahre bedanken, er ist mir stets mit Rat und Tat zur Seite gestanden. Ebenso danke ich meinen Kolleginnen und Kollegen, mit denen in den fünf Jahren ein bereichender Austausch zu einer Vielzahl von Themen stattgefunden hat und viele schöne Momente, insbesondere auf gemeinsamen Konferenzen, entstanden sind. Weiters wurden mir zwei horizonterweiternde Auslandsaufenthalte in Lille und Málaga ermöglicht, wofür ich mich bei meinen jeweiligen Gastgebern El-Ghazali Talbi und Francisco Chicano herzlich bedanken möchte. Nicht weniger Dank gebührt meiner Familie, meinen Eltern Brigitte & Konrad, meiner Schwester Anja, und meinen langjährigen Freunden Klaus & Max, die mir jederzeit unterstützend zur Seite gestanden sind.

Acknowledgements

First and foremost, I would like to thank my supervisor Günther Raidl for the great cooperation over all the years. He has always supported me with advice and action, inspired me with his ideas, and provided me with many interesting opportunities. Likewise, I am thankful to my colleagues, with whom I had many fruitful discussions over the five years and numerous nice moments in particular at joint conferences. Two horizon-broadening stays abroad in Lille and Málaga were made possible, for which I am sincerely grateful to my respective hosts, El-Ghazali Talbi und Francisco Chicano. No less thanks are due to my family, my parents Brigitte & Konrad and my sister Anja, and my longtime friends Max & Klaus, who have supported me at any time.

In addition to my supervisor, I want to especially thank Thomas Jatschka for taking the time to read the first version of my thesis and providing me with corrections and much-appreciated comments. Furthermore, I want to express my gratitude to Christian Blum and Sophie Parragh for taking the time to review this thesis and providing me with valuable feedback and also to Luca Di Gaspero and Nysret Musliu for being part of my evaluation committee. Thanks are also due to my further co-authors Adrian Bracher, Jan Gmys, Matthias Horn, Nouredine Melab, Bernhard Neumann, Giulio Pace, Stephan Teuschl, and also to the anonymous reviewers of our publications. I would also like to thank our industry partners for the interesting and challenging joint projects.

I am thankful for the partial funding of this thesis by the Doctoral Program "Vienna Graduate School on Computational Optimization", Austrian Science Foundation (FWF) Project No. W1260-N35, TU Wien's KUWI grant, and, furthermore, for the computational resources provided by the Algorithms & Complexity Group's compute cluster and the Grid'5000[1] testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations to enable the experiments presented in this thesis.

[1] https://www.grid5000.fr

Kurzfassung

Das Modellieren des allgemeinen Lösens von Problemen als Suche in einem Zustandsraum ist eine wohlbekannte Abstraktion. Eine komplexe Aufgabe wird dabei in eine wohldefinierte Modellwelt transferiert, in der das Finden von und Argumentieren über Algorithmen ungleich einfacher ist. Ein Hauptmerkmal ist die Zerlegung einer Lösung für ein Problem in eine Abfolge von Aktionen, wobei diese gewisse Eigenschaften haben soll. Es ist ein mächtiges Paradigma welches u.A. in den Bereichen künstliche Intelligenz, kombinatorische Optimierung und stochastischer Planung zum Einsatz kommt.

Das Ziel dieser Arbeit ist das Voranbringen der Zustandsraumsuche angewandt auf statische kombinatorische Optimierungsprobleme und stochastische Entscheidungsprobleme, in der eine Reihe von abhängigen Unterproblemen gelöst werden muss. Wenngleich schon gut entwickelte Theorien und viele exakte und heuristische Lösungsansätze existieren, gibt es immer noch genügend Raum zur Weiterentwicklung. Einerseits setzt die kombinatorische Explosion praktischen Lösungsverfahren Grenzen, welche immer wieder ein Stück versetzt werden wollen. Andererseits haben dynamische und stochastische Probleme vermehrt an Bedeutung gewonnen. Datengetriebene Ansätze und Hochleistungs-Computer bieten neue Werkzeuge, die weiter mit bestehenden Methoden kombiniert werden möchten.

Im ersten Teil dieser Arbeit konzentrieren wir uns auf wohlbekannte statische Optimierungsprobleme. Wir beginnen damit eine neue Zustandsraum-Formulierung für ein praktisch besonders schwieriges Problem aufzustellen, für das *Traveling Tournament Problem* (TTP). Wir schlagen heuristische Lösungsverfahren basierend auf *Beam Search* vor, einem heuristischen Suchalgorithmus. Wir untersuchen unterschiedliche, auf unteren Schranken basierende Evaluierungsfuntionen und zeigen wie diese randomisiert werden können, um die Suche erfolgreich zu diversifizieren und zu parallelisieren.

Mit der steigenden Verfügbarkeit von Mehrprozessor-Systemen und gleichzeitigem geringeren Wachstum von Einzel-Prozessor Durchsatz im letzten Jahrzent, gewinnt Parallelisierung noch mehr an Bedeutung. Inspiriert durch die erfolgreiche Anwendung auf das TTP, erweitern wir unseren Ansatz zu einem generellen parallelen *Beam Search Framework*, entwickelt in der Programmiersprache Julia, für kombinatorische Optimierungsprobleme. Angewandt auf das TTP, das *Permutation Flowshop Problem* (PFSP) und das *Maximum Independent Set Problem* (MISP), zeigen wir die erhebliche Verkürzung der Laufzeiten durch mittel bis hohe parallele Effizienz für ausreichend große Probleminstanzen, und finden viele neue beste Lösungen für TTP Benchmark-Instanzen.

Im zweiten Teil befassen wir uns mit dynamischen und stochastischen Problemen. Zeit betritt als zentraler Parameter die Bühne und Information wird uns Stück für Stück serviert, wobei wir stochastische Informationen in Form von Wahrscheinlichkeitsverteilungen von Beginn an zur Verfügung haben. Aktionen sind nicht mehr deterministisch und der Ablauf im Zustandsraum kann oft über einen *Markov decision process* (MDP) modelliert werden, in welchem Aktionen und Zustandsübergänge probabilistischer Natur sind. Das allgemeine Ziel ist das Finden sogenannter *Policies*, welche die Entscheidungen determinieren und gut für eine Instanzklasse eines Problems funktionieren sollen.

Als exemplarisches, schwieriges stochastisches Entscheidungsproblem, führen wir ein *Lieferung-binnen-einer-Stunde Problem* ein, welches von einem Online-Supermarkt in Wien stammt, der Lieferungen innerhalb einer Stunde verspricht. Im Kern ist es ein dynamisches und stochastisches Tourenplanungsproblem mit einem dynamischen Schichtplanungsaspekt, da wir auch die möglichen Überstunden der Fahrer managen müssen. Das Ziel ist die automatische Tourenplanung und Entsendung der Lieferflotte über einen Tag hinweg, um auf die eingehenden Bestellungen bestmöglich zu reagieren. Das Primärziel ist Kundenzufriedenheit, wobei die Lieferkosten als Sekundärziel minimiert werden sollen.

Ob der Problemkomplexität, führen wir einen Zerlegungsansatz durch. Für die statische Tourenplanung schlagen wir eine wohlbekannte *Adaptive Large Neighborhood Search* (ALNS) vor. Zur dynamischen Schichtplanung verwenden wir einen Dual-Horizont Ansatz. Dabei wird vor jeder Entscheidung ein vereinfachtes Unterproblem für die Zukunft gelöst, um die benötigten Überstunden der Fahrerinnen abzuschätzen. Als Wartestrategie für Routen schlagen wir eine Heuristik vor, welche Routen, die bereits effizient genug sind, früher startet als ineffiziente. Wir evaluieren unsere Ansätze auf repräsentativen Instanzen und zeigen, dass diese einen balancierenden Effekt auf die drei Dimensionen Verspätungen, Fahrtkosten und Überstunden haben.

Die Verteilung der mittleren Lieferdauer, welche wir auch Routen-Performance nennen ändert sich mit der Zeit und hängt stark von der Anzahl der Bestellungen – der aktuellen Last – und der Verkehrssituation ab. Eine Abschätzung bzw. Vorhersage dieser Kenngröße ist sehr wichtig für verschiedene Aspekte wie der statischen Schichtplanung im Vorhinein, der dynamischen Schichtplanung am selben Tag und der vorher erwähnten Wartestrategie. Zu diesem Zwecke untersuchen wir unterschiedliche Vorhersagemodelle basierend auf überwachtem Lernen auf simulierten und echten Daten.

Der klassische Ansatz aus der Literatur ist eine Stichprobe von Szenarien der nahen Zukunft zu ziehen und die sich daraus ergebenden statischen Tourenplanungsprobleme bestehend aus echten und virtuellen Bestellungen zu lösen und daraus eine Konsensus-Entscheidung abzuleiten. Sein größter Nachteil ist der Berechnungs-Aufwand, da die Stichprobe ausreichend groß sein muss. Wir schlagen daher eine günstigere Methode zur Nachahmung des Stichproben-Ansatzes vor, bei dem in einer *offline* Trainings-Phase die stochastische Information als eine approximative Bewertungsfunktion direkt in die Zielfunktion transferiert wird. Dadurch bleiben die Berechnung für die kurzfristig benötigten *online* Entscheidung schnell, aber berücksichtigen auch implizit die nähere Zukunft und nicht nur den kurzsichtigen Ertrag.

Abstract

State-space search is a well-known abstraction for general problem solving. Its merit is the transformation of a complex task into a well-behaved model world in which it is easier to devise algorithms and to reason about them. A main feature is the decomposition of a problem's solution into a sequence of actions, where we are interested in finding a sequence with some desired properties. It is a powerful paradigm used for instance in artificial intelligence, planning, combinatorial optimization, and stochastic decision making.

In this work, we aim to advance state-space search for static combinatorial optimization and for stochastic decision making, where a series of dependent subproblems has to be solved. Well-developed theories and many exact and heuristic solution approaches to tackle this broad class of problems already exist. Still, there is room left for improvement due to the combinatorial explosion with growing problem size and the shift from static towards dynamic and stochastic problem variants, creating new challenges. With the recent rise of data-driven approaches based on machine learning methods and the increase of high performance computing resources, we have new tools at our disposal to be incorporated into existing solution approaches to face these challenges.

In the first part of this thesis, we focus on state-space search for static combinatorial optimization problems. We start by introducing a novel state-space formulation to a particularly challenging benchmark problem from sports league scheduling, the traveling tournament problem (TTP). We propose to solve it heuristically with different *beam search* variants, a heuristic search algorithm, extending lower bound calculation methods from the literature to guide the search with randomization to diversify the search. This allows us to find several new best feasible solutions for long-standing difficult benchmark instances. Additionally, we study bounded suboptimal weighted A* search behavior.

With the increased availability of many-core system clusters and the reduced slope in single-threaded performance increase over the last decade, parallelization in solvers becomes even more of a topic. Inspired by the success on the TTP, we propose and implement a general *parallel beam search* framework in Julia for combinatorial optimization problems. The natural approach is to split each layer's workload and distribute it among multiple cores. We demonstrate substantial speedups on the TTP, the permutation flowshop problem (PFSP), and the maximum independent set problem (MISP), with medium to high parallel efficiency for sufficiently large problem instances.

In the second part of the thesis, we move from static to dynamic and stochastic optimization problems. Time enters as a parameter and information is revealed to us bit by bit and we have only stochastic information in form of probability distributions available upfront. Actions are not deterministic anymore and the overall problem is best modeled as Markov decision process (MDP), where actions and state transitions are probabilistic. The general goal is to devise policies and decision rules, using handcrafted heuristics or approaches based on approximate dynamic programming (ADP).

As a challenging stochastic decision-making problem, we introduce a real-world same-hour delivery problem originating from an online supermarket in Vienna, which promises to deliver orders within the hour. The task is to automatically dispatch a fleet of vehicles by planning their delivery routes and potential overtime to react to dynamically arriving orders by stochastic customers. The primary objective is to maximize customer satisfaction while minimizing the real delivery costs as a secondary objective.

Due to the problem complexity we employ a decomposition approach. For the static route optimization part, we propose and tune a well-known *adaptive large neighborhood search* (ALNS). To dynamically plan shifts, we propose a dual-horizon approach, where at each decision epoch a simplified subproblem is solved in a sampling horizon to derive desired overtime for the drivers, which is then fed back into a modified objective function for the ALNS. As a waiting strategy, we propose a simple yet effective heuristic that lets sufficiently efficient routes start earlier than inefficient ones. We compare our approaches on different real-world inspired artificial instances and show the balancing effect on the three dimensions tardiness, travel times, and driver overtime.

The distribution of mean order delivery times, which we also call route or driver performance, changes over time, depending mostly on the number of orders available and the traffic. Having an estimate for this performance value is important for other decisions, like the static shift planning upfront, the dynamic shift planning during the day, and the aforementioned waiting strategies. To this end, we study different predictors trained by supervised learning on simulated and historical real-world data which can be used in the online decision making.

A classical approach from the literature is to sample multiple scenarios in a short horizon and solve static vehicle routing problems combining real and sampled orders. The resulting solutions are then used to derive a consensus decision taking the stochastic knowledge into account, unlike a myopic approach. The main drawback is its high computational effort since sufficiently many scenarios have to be evaluated. In the two final chapters of this thesis, we, therefore, build on the approximate dynamic programming paradigm to emulate the behavior of scenario sampling driven policies by training an approximation function offline, which is then incorporated into an augmented objective function for the ALNS. The online computation is then still sufficiently fast for near real-time decisions and also makes use of the stochastic information considering the value decisions instead of only considering the current reward.

Contents

CHAPTER 1

Introduction

Search is an essential technique in general problem solving applied both by humans and machines. It is a mental or mechanical exploration of a model world as to find a desirable sequence of actions to either come closer or achieve a certain goal. In computer science, it prominently appears in the fields of artificial intelligence, planning, combinatorial games, and optimization. For instance, multiple agents with initial positions on a grid have to find a path to their goal states without colliding while minimizing the time needed for the last one to be in its final state [140]. Newell and Simon in their epic book from 1972 [112] "Human problem solving" argue that searching in a so-called problem space is also how parts of human thinking work.

The focus of this thesis is on combinatorial optimization problems modeled as search problems in state spaces. This well-known paradigm amounts to decomposing the solving process of problem instances into a sequence of decisions associated with costs and using local information to guide this process or to exclude unpromising decisions early. For static optimization problems, the transitions between the states are deterministic and we have complete information from the start. For dynamic and stochastic optimization problems time appears as a parameter and we operate within a non-deterministic setting.

Combinatorial optimization problems have the distinguishing feature that for a problem instance the number of solutions is finite and could in principle be enumerated to find a best solution according to defined criteria, e.g., finding the shortest path from a start to a goal vertex or finding a minimum cost Hamiltonian cycle in a graph. Still, the solving faces the challenges of combinatorial explosion, i.e., the rapid growth of the way elements can be arranged or selected with the instance size, and the curse of dimensionality, the growth of the number of possible value combinations when adding finite-domain variables to the problem—new dimensions. Most interesting problems are often $\mathcal{NP}$-hard, i.e., are widely believed to have in general a worst-case runtime which is at least exponential in the input size, unless $\mathcal{P} = \mathcal{NP}$.

Classical complexity theory goes back to the late 60s and 70s [31, 63]. Since then the concepts of fixed parameter tractability and parameterized complexity [42, 34] have emerged, where the complexity of instance classes can be captured and isolated into a certain structure with parameter (e.g., treewidth of a graph), and instances become tractable again if this parameter is bounded.

Another natural way to practically attack large problem instances is to drop the need for optimality, i.e., it suffices or even is required to find high-quality solutions in reasonable time without seeking to provide an optimality guarantee. This is then called heuristic optimization in general or heuristic search when we emphasize the search aspect in problem solving, our main research field.

In Chapter 3 of this thesis, we begin with static optimization problems, where we make contributions to advancing state-space search based algorithms. To this end, we consider the traveling tournament problem (TTP) as a prototypical benchmark problem introduced by Easton, Nemhauser, and Trick [46]. The goal is to construct a double round robin tournament for an even number of teams so that their total travel distance is minimized (teams go directly from venue to venue) with a maximum number of home or away games in a row and no games between two teams back-to-back. It is a challenging benchmark problem for exact and heuristic methods combining difficult feasibility and optimality aspects and rapid growth of the search space with many symmetries in the search space.

We propose a novel state-space formulation on which we apply a randomized beam search algorithm, for which we provide a thorough empirical study using different guidance functions and compare with the so-far state-of-the-art simulated annealing approach. We observe that a combination of beam search and a fast local search comes close to so-far best-found solutions and finds new best solutions for several long-standing instances. Furthermore, we show the behavior of bounded suboptimal weighted A* search on the TTP's state graph over different optimality guarantees, and how the number of node expansions can be reduced by beneficial variable ordering and duplicate detection.

The recent increase in computational resources is largely attributed to parallelization on CPU core and GPU level, as the clock frequencies of CPUs have reached a natural physical limit. To address this, we generalize our TTP beam search algorithm to a general, parallel beam search framework for optimization in Chapter 4, where we focus on shared-memory systems with many cores and uniform memory. We demonstrate large speedups and medium to high parallel efficiency on three NP-hard combinatorial optimization problems: For the permutation flowshop problem (PFSP), said TTP, and the maximum independent set problem (MISP). For the TTP we perform large-scale runs, by which we could find many new feasible solutions for difficult benchmark instances from the literature and are on par with the state of the art in terms of the mean solution quality.

Afterwards, we move on to dynamic and stochastic optimization, where we focus on same-day delivery problem variants [162] originating from a corporation with an online supermarket in Vienna, which promises to deliver goods within the same hour. These

kinds of problems are dynamic and stochastic vehicle routing problems [126] in which orders of customers, which are only revealed over time and unknown at the beginning of the planning horizon, have to be delivered within short deadlines given a fleet of vehicles. The task is to automatize the process of deciding which orders should be delivered when and by which driver, so that we first maximize customer satisfaction and second minimize the company's expected costs. Furthermore, we are also given a spatiotemporal distribution of the load pattern over the day and the question is how to incorporate this stochastic information into our near real-time decision making process.

In Chapter 5, we introduce the general problem and a solution approach decomposing the different aspects of this complex problem. The routing and dynamic shift planning is based on an adaptive large neighborhood search (ALNS) and a dual horizon optimization [109]. The latter solves a simplified problem for a larger horizon into the future and feeds this information into the shorter horizon optimization done by the ALNS to subsequently derive near real-time decisions. Another important decision is when to start routes [86, 163]. We propose a simple yet effective heuristic that starts more efficient routes earlier and still inefficient routes later. Combined to a whole decision making entity, we compare our approach on different real-world inspired artificial instances with baseline strategies and show the balancing effect regarding the three objective dimensions tardiness, travel times, and driver overtime, when considered as multiple objectives.

Estimating the mean order delivery time of drivers throughout the day depending on the load pattern and traffic is an important parameter for the decision making, e.g., to anticipate how many drivers we need or to estimate the relative performance of the routes. In Chapter 6 we study different machine learning models to predict the performance of drivers, which mainly depends on traffic and load, given real-world data from Vienna and discuss how it can be used in the optimization. While for small-load situations the prediction remains challenging due to high variance, for medium and high-load situations effective unbiased predictors could be trained.

One standard method from the literature is the multiple scenario sampling approach (MSA) with consensus function [11, 162], where within a sampling horizon a number of scenarios are sampled, solved as offline problems, and then the obtained scenario plans are used to derive the next decision. A drawback is that this approach is computationally very demanding and therefore often prohibitive for real-time use. In the last two Chapters 7 and 8 of the thesis' main body we discuss supervised learning approaches to train either a surrogate function or a value function [158, 89] for the routing decisions of same-hour delivery in a computationally demanding offline phase, which is then incorporated in the objective function for the time-critical online optimization. We show that the routing behavior of MSA, which reduces travel duration and tardiness substantially compared to myopic policies, can be successfully emulated using this surrogate function approach while still allowing near-realtime decisions.

To summarize, the main contributions of this thesis are:

- State-space formulation and randomized beam search for the challenging TTP.

- Thorough computational study on difficult TTP benchmark instances from the literature of said beam search compared with weighted A* search and state-of-the-art simulated annealing.
- Extension to a general, parallel beam search framework for combinatorial optimization with large speedups and medium/high parallel efficiency over a range of exemplary problems with different properties.
- A dual-horizon approach for same-hour problems with shift flexibility, a challenging dynamic vehicle routing problem with stochastic customers.
- A supervised learning approach to predict simulated and real-world mean order delivery times for same-hour delivery problems.
- A supervised learning approach to estimate value functions/surrogate functions to be used in the point-in-time optimization to account for the dynamic and stochastic aspects of problems.

We now give an overview of the structure of this thesis and our related previous publications upon which it is based.

1.1 Structure of the Thesis

The next chapter is dedicated to the methodology, where we concisely discuss relevant aspects of heuristic optimization, state space search, shared-memory parallelization, Markov decision processes, approximate dynamic programming, and supervised learning for regression problems.

The main body of the thesis can be seen as split into two parts. The first part deals with state-space search for static combinatorial optimization problems. In particular, we focus on the heuristic method *beam search*, which we apply to the famously challenging traveling tournament problem in Chapter 3. Driven by its computational demand, we derive a general, parallel beam search framework for combinatorial optimization problems in Chapter 4 which we study on an exemplary set of combinatorial optimization problems with different properties. The work on beam search for the TTP was first presented at the EvoCOP 2020 conference and published in its proceedings. Afterwards, a subsequent invited extended journal version was published in the Evolutionary Computation Journal:

> Nikolaus Frohner, Bernhard Neumann, and Günther R Raidl. A beam search approach to the traveling tournament problem. In *Evolutionary Computation in Combinatorial Optimization – 20th European Conference, EvoCOP 2020*, volume 12102 of *LNCS*, pages 67–82. Springer, 2020
>
> Nikolaus Frohner, Bernhard Neumann, Giulio Pace, and Günther R Raidl. Approaching the traveling tournament problem with randomized beam search. *Evolutionary Computation Journal*, 2022. in press

The concepts behind the parallel beam search framework were published as extended abstract in the proceedings and presented in the poster session at the SoCS 2022:

> Nikolaus Frohner, Jan Gmys, Nouredine Melab, Günther R Raidl, and El-Ghazali Talbi. Parallel beam search for combinatorial optimization (extended abstract). *International Symposium on Combinatorial Search*, 15(1):273–275, 2022

The full version of the paper was presented at the 51st International Conference on Parallel Processing (ICPP) in the Parallel and Distributed Algorithms for Decision Sciences (PDADS) workshop and published in the corresponding proceedings:

> Nikolaus Frohner, Jan Gmys, Nouredine Melab, Günther R Raidl, and El-Ghazali Talbi. Parallel beam search for combinatorial optimization. In *51th International Conference on Parallel Processing Workshop*, ICPP Workshops '22. Association for Computing Machinery, 2022

Before we shifted our focus to heuristic methods, in particular beam search, we also worked on improving dual bounds using relaxed binary decision diagrams for combinatorial optimization problems (not included in this thesis), in particular on the MISP:

> Nikolaus Frohner and Günther R Raidl. Towards improving merging heuristics for binary decision diagrams. In *Proceedings of LION 13 – 13th International Conference on Learning and Intelligent Optimization*, volume 11968 of *LNCS*, pages 30–45. Springer, 2019

> Nikolaus Frohner and Günther R Raidl. Merging quality estimation for binary decision diagrams with binary classifiers. In *Machine Learning, Optimization, and Data Science – 5th International Conference, LOD 2019*, volume 11943 of *LNCS*, pages 445–457. Springer, 2019

In another separate project not presented here, we combined constraint programming with metaheuristics to tackle a real-world employee scheduling problem:

> Nikolaus Frohner, Stephan Teuschl, and Günther R Raidl. Casual employee scheduling with constraint programming and metaheuristics. In *Computer Aided Systems Theory – EUROCAST 2019*, volume 12013 of *LNCS*, pages 279–287. Springer, 2020

In the second part of the thesis, time is introduced as a parameter moving to dynamic optimization problems. In Chapter 5 we introduce a new same-day delivery problem variant originating from the real world, a challenging dynamic and stochastic vehicle routing problem. It is based on the work published at the PATAT 2020 (due to COVID-19 postponed twice, became PATAT 2022):

Nikolaus Frohner and Günther R Raidl. A double-horizon approach to a purely dynamic and stochastic vehicle routing problem with delivery deadlines and shift flexibility. In *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling - PATAT 2021: Volume I*, 2020

The subsequent chapters deal with further aspects and variants of this problem class. In Chapter 6 we study different machine learning models on real-world data to predict the performance of drivers in form of the mean order delivery time for a given time and day. The work has been presented at the ISM 2020 and has been published in the corresponding open access proceedings:

Nikolaus Frohner, Matthias Horn, and Günther R Raidl. Route duration prediction in a stochastic and dynamic vehicle routing problem with short delivery deadlines. In *Proceedings of the 2nd International Conference on Industry 4.0 and Smart Manufacturing (ISM 2020)*, volume 180 of *Procedia Computer Science*, pages 366–370. Elsevier, 2021

In Chapters 7 and 8, we present approaches to replace the computationally heavy sampling method with offline learning of a value function that is used in the online optimization augmenting the myopic objective function to account for stochastic and dynamic aspects of the problem. These works have been presented at the CPAIOR 2021 and the EUROCAST 2022 conferences, respectively, and published in the corresponding proceedings:[1]

Adrian Bracher, Nikolaus Frohner, and Günther R Raidl. Learning surrogate functions for the short-horizon planning in same-day delivery problems. In *17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR'21)*, volume 12735 of *LNCS*, pages 283–298. Springer, 2021

Nikolaus Frohner and Günther R Raidl. Learning value functions for same-day delivery problems. In *Computer Aided Systems Theory – EUROCAST 2022*, LNCS. Springer, 2022. accepted

Where applicable, relevant extensions to the publications in this thesis are discussed at the end of each chapter's abstract. Corresponding new sections/paragraphs are marked with a dagger† symbol.

[1] publication of the EUROCAST paper has been accepted but delayed to the next edition 2024

CHAPTER 2

Methodology

In this chapter, we introduce the main methods used to approach our static and dynamic optimization problems. We begin by giving an overview on heuristic optimization in general, which deals with finding high-quality solutions to practically relevant instance classes of optimization problems in reasonable time, while not necessarily having an optimality guarantee. We subsequently focus on state space search, a well-known paradigm modeling a problem as—either heuristic or exact—search for a goal state in a state graph starting from an initial state. It is well known in AI and planning but also applicable to static combinatorial optimization problems. Apart from devising stronger heuristics to reduce the number of search nodes to be expanded or to improve the solution quality given the same number of nodes searched, increasing the throughput by parallelization is another method to speed up corresponding search algorithms. We briefly discuss data parallelism on multi-core shared memory systems which is the main parallelization tool in this thesis.

So far, we have focused on static optimization which results in deterministic transitions on the state graph. In the second main part of this thesis, we discuss same-day delivery problems, which are dynamic vehicle routing problems with stochastic customers. Time and uncertainties enter our scope and information is only revealed dynamically, such that we end up with a sequence of linked optimization problems together with stochastic decision making. Our state space is now best modeled as Markov decision process, which we discuss together with approximate dynamic programming. Its transitions are probabilistic and we can only reason about statistics of solution approaches (e.g., expected costs) by considering many realizations from an instance class. One method to tackle such problems is to perform roll-outs at decision epochs and derive an actual decision to perform. Such roll-outs are often computationally expensive and we therefore aim at learning auxiliary functions in a training phase to perform informed online decisions more quickly. To this end, we discuss supervised learning for regression problems, to use suitably trained models as estimators and predictors in our dynamic problems.

2.1 Heuristic Optimization

Many interesting problems in combinatorial optimization are difficult to solve to proven optimality in practice, where even methods with clever reasoning struggle with the enormous size of the search space of sufficiently large instances. The concrete quantification of the practical limit for the instance size is dependent on the considered problem, whether we focus on particular instance classes with a certain exploitable structure reducing complexity, the currently available state-of-the-art solution approaches and implementations, computing resources, potential parallelization, and how much time we grant ourselves to wait for the solution.

As illustration, consider the traveling salesperson problem (TSP), where a non-trivial problem instance with 85 900 vertices based on circuit design was solved within hundreds of CPU years on a cluster in the 2000s to proven optimality using a highly sophisticated branch-and-cut algorithm implemented in the Concorde[1] solver—the journey of its genesis is presented in the book by Applegate, Bixby, Chvátal, and Cook [6]. On the other side of the spectrum lies the traveling tournament problem (TTP), where a double round robin (DRR) tournament with n teams has to be scheduled while minimizing the travel distance over all teams subject to constraints on the schedule. Instances with as little as ten teams could be solved within one to two CPU years to optimality with a sophisticated state-of-art iterative deepening A* approach [155], while instances with 12 teams and more still remain unsolved in an exact fashion up to this date. What many interesting optimization problems share is that sooner or later the combinatorial explosion kicks in and that any exact algorithm is believed to have at least exponential runtime in the size of the input, unless $\mathcal{P} = \mathcal{NP}$. We formalize combinatorial optimization problems as follows:

Definition 2.1. A combinatorial optimization problem (COP) is defined by a set of instances $\mathcal{C}$, where a given problem instance $C \in \mathcal{C}, C = \langle S, f \rangle$ has a finite solution space S together with an objective function $f\colon S \to \mathbb{R}$ to be minimized.[2] The goal of the corresponding search problem is to find a globally optimal solution x^*, i.e., for which the objective value $z^* = f(x^*) \leq f(x')\ \forall x' \in S$. (Papadimitriou and Steiglitz [114])

In heuristic optimization, we make a trade-off. We sacrifice solution quality in return for shorter runtime. On a high level, this is done by drastically reducing the considered search space, since we do not have the burden of providing optimality guarantees anymore. We still need to balance between *diversification*, to cover much of it on a coarse level and not to miss interesting parts, and *intensification*, to more closely look into promising regions to find high-quality solutions. *Metaheuristics*, like simulated annealing (SA, [91]), variable neighborhood search (VNS, [110]), tabu search (TS, [69]), genetic algorithms (GA [82]), and many more, provide algorithmic templates related to this dilemma applicable on a wide variety of problems. A detailed compilation of this weaponry is

[1] `https://www.math.uwaterloo.ca/tsp/concorde.html`

[2] Maximization can be converted into a minimization by applying the transformation $f(x) \to -f(x)$.

presented in the Handbook of Metaheuristics [66]. Suitable problem-specific methods and operators concerning the *exploration* of the search space and the *exploitation* of promising information have to be devised for a successful solution approach. Proper algorithmic tuning is necessary to avoid premature convergence to unsatisfactory solutions.

To categorize heuristics, we mainly distinguish them along the following two dimensions.

- Constructive methods to create a solution from scratch or complete a partial solution vs. local improvement methods, which seek to improve complete solutions by moving around in the solution space—this distinction can sometimes be blurry,
- single-trajectory approaches, traversing the search space by keeping only one current solution vs. population-based approaches, where we keep a set of solutions spread across the search space with potential interaction between them ("the sum is more than its parts").

Different types of (meta-)heuristics are meant to work together, we often encounter representatives of each category entangled in complete and in particular competitive solution approaches. For example, in a genetic algorithm (a population-based method) we need to create solutions for a diverse initial population and may apply a single-trajectory local improvement algorithm on each individual before moving to the next generation. More sophisticated combinations of metaheuristics and other search methods are discussed in the book by Blum and Raidl [18], under the umbrella of *hybrid metaheuristics*.

Constructive methods are discussed in greater detail in the next section in the framework of state-space search. Beam search [105, 113] will be the star of the first part of the main body of this thesis, which is rather simple in its design yet proven to be an empirically effective search algorithm. It follows a varying bundle of promising partial solutions in parallel on a graph that represents the set of partial solutions (also called construction graph) and returns the best found complete solution if it has found one. Therefore, it clearly falls in the category of constructive heuristics and additionally can be seen as population-based, in the sense of keeping a population of *partial* solutions. In the remainder of this section, we first discuss the basics of neighborhood-based methods and then focus on two neighborhood-based metaheuristics we will build on in this thesis, namely simulated annealing (SA) and adaptive large neighborhood search (ALNS).

Local search. The classic neighborhood-based improvement heuristic is *local search*, often part of other, more sophisticated metaheuristic approaches. The key element is a *neighborhood structure* N. It maps from a solution $x \in S$ to a set of neighbors $N(x) \subset S$, the *neighborhood*. This allows us to define *local optima*, i.e., all solutions $\tilde{x} \in S$, for which $f(\tilde{x}) \leq f(x') \; \forall x' \in N(\tilde{x})$. For a solution to be globally optimal, it also has to be locally optimal, which is the motivation behind the local search procedure: Starting from an initial solution x_0, it finds a local optimum by maintaining a current solution $x \in S$ with objective value $f(x)$ and performing small improving steps in the

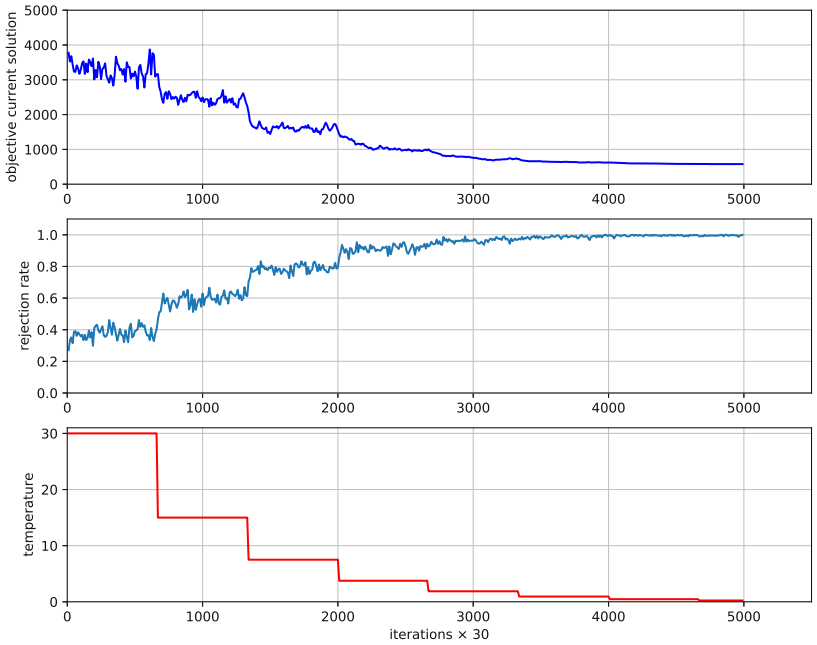


Figure 2.1: Exemplary SA run on a TSP instance with 2-opt neighborhood showing the objective value of the current solution, the rejection rate, and the temperature over the iterations.

solution space. A well-known neighborhood structure is 2-opt for the TSP which seeks to remove path crossings by reversing and reconnecting path segments. Another algorithmic component is the *step function*, which defines how these steps in the neighborhoods are performed. We distinguish between best-improvement, which selects an $x' \in N(x)$ for which $|f(x) - f(x')|$ is a maximum over $N(x)$, next-improvement, which iterates over all neighbors in a defined order and takes the first x' that improves the objective value, and random-improvement, which randomly samples neighbors for a number of iterations and takes the first x' that improves the objective value. The search terminates at a (probabilistic) local optimum $\tilde{x}$, when no strict improvement has been found. One key goal of single-trajectory based metaheuristics is to escape such local optima to cover a wider area of the search space, in the hope to find a local optimum that is also (or close to) globally optimal, as done by, e.g., iterated local search (ILS) [104] or VNS [110].

Simulated annealing. The name originally derives from metallurgy where by repeatedly heating and cooling metal, crystal defects are reduced. The method itself is related to the Metropolis-Hastings algorithm [107, 80] to sample from a distribution via a guided random walk in a sample space, where drawing samples directly is difficult. The original application comes from statistical mechanics where we consider states with different energies, on which their probability to occur depends. In thermal equilibrium, lower energy states have a higher probability of being occupied than higher energy states.

In the adaption for optimization as proposed by Kirkpatrick et al. [91], the energy is

Algorithm 2.1: Simulated Annealing.

Input: initial solution x_0, neighborhood structure N, initial temperature T_0, equilibrium iterations N^{eq}, cooling schedule function τ
Output: best found solution x^*

$x^* \leftarrow x_0, x \leftarrow x_0, T \leftarrow T_0$;
while *Termination-Criterion not met* **do**
  foreach $i \in \{1, \ldots, N^{\text{eq}}\}$ **do**
    $x' \leftarrow$ sample from $N(x)$;
    if $f(x') < f(x)$ **then**
      $x \leftarrow x'$;
      if $f(x') < f(x^*)$ **then**
        $x^* \leftarrow x$;
      end
    else
      $q \leftarrow$ sample from $[0, 1]$;
      if $q < e^{-\frac{|f(x)-f(x')|}{T}}$ **then**
        $x \leftarrow x'$;
      end
  end
  $T \leftarrow \tau(T)$;
end
return x^*;

replaced by the objective value, and the goal is to guide the sampling by the Boltzmann distribution $\propto e^{-f(x)/T}$. SA performs a guided random walk by iterative sampling from the neighborhood $N(x)$ of the current solution x and conditionally moving to the sampled neighbor. In each iteration, a neighbor x' is proposed and its acceptance is probabilistic (denoted as $\mathbb{P}(x \to x')$) following the famous Metropolis criterion [107]. Improving solutions are always accepted, otherwise it depends on the distance in objective value to the current solution and the current temperature:

$$\mathbb{P}(x \to x') = e^{-\frac{|f(x)-f(x')|}{T}} \tag{2.1}$$

Keeping the objective distance fixed, the higher the temperature, the more likely we will move to a neighbor, and the focus is still on diversification. What actually happens is that for higher temperatures the probability differences related to objective value differences are smeared and become similar ($e^{-f(x)/T}$ is flat in its tail, the derivative is almost zero), while for low temperatures the sample space becomes pronounced (the exponential close to zero is steep). The initial temperature T_0 is set to have a rather high acceptance rate and therefore to be more explorative in the beginning of the search. Over the time of the search, we decrease T following a cooling schedule and move to the intensification

regime. A pseudo-code of SA is shown in Algorithm 2.1. A common cooling schedule is geometric cooling where T is multiplied by $\beta < 1$ after N^{eq} equilibrium iterations on a temperature level have been performed. Additionally, reheating to a higher temperature can be performed when we have reached a low temperature/high rejection rate, since at some point we may get probabilistically stuck in a local optimum and search effort goes to waste.

The course of the objective value of the current solution, the rejection rate of the sampled neighbors, and the temperature on an example SA run on a TSP instance with 2-opt neighborhood are depicted in Figure 2.1. The temperature tunes the balance between diversification (high temperature, high acceptance rate) and intensification (low temperature, low acceptance rate), so it is versatile in this sense. Another advantage is that when enumerating all neighbors of x is expensive, a sample-accept-reject procedure may be fast and allows traversing the search space more quickly.

If feasible solutions and neighbors cannot easily be enumerated or the natural solution encoding has many infeasible solutions, it makes sense to also allow infeasible solutions and penalize the "degree of infeasibility" $\gamma(x)$. The objective could then be extended to a sum $f(x) + w\gamma(x)$, where w can be made time-dependent and oscillating to balance the search between infeasible and feasible regions and allow easier traversals of infeasible parts of the search space. This approach is called strategic oscillation and was originally proposed for tabu search, see for instance the book by Glover and Laguna [70].

Apart from an efficient implementation to achieve a high solution evaluation throughput, parameter tuning is necessary for simulated annealing to be effective and efficient. For the details of a recent example application to a difficult and highly constrained sports league scheduling problem, see Rosati et al. [128], where a multi-neighborhood SA is used in multiple properly tuned search phases to achieve highly competitive results.

(Adaptive) large neighborhood search. As opposed to classic local search, large neighborhood search (LNS) considers neighborhoods that are too large for naive enumeration. Instead, the related sub-problem of finding a best or promising neighbor is solved efficiently. LNS was originally introduced by Shaw [136] in the context of constraint programming, where the search was implemented by iteratively destroying and reconstructing parts of the solution, i.e., where a part of the solution remains fixed and all the possible feasible completions correspond to the neighborhood. The destruction and recreation are performed by heuristic operators with stochastic elements. The general idea is to search neighborhoods of exponential size to be able to leave unpromising regions of the search space more quickly.

Ropke and Pisinger [127, 117] have extended the idea to Adaptive LNS (ALNS) in the context of vehicle routing problems. Instead of using only one operator pair, multiple potentially parameterized destroy and repair (we use the different names so far interchangeably) operators are at our disposal and their usage frequency is learned online by adapting corresponding weights depending on their success so far.

Algorithm 2.2: Adaptive Large Neighborhood Search.

Input: initial solution x_0, set of destroy operators $\boldsymbol{\Omega}^-$ and repair operators $\boldsymbol{\Omega}^+$, initial temperature T_0, equilibrium iterations N^{eq}, cooling schedule function τ, weight decay λ

Output: best found solution x^*

$x^* \leftarrow x_0, x \leftarrow x, T \leftarrow T_0$;
initialize weights vectors $\boldsymbol{\omega}^-, \boldsymbol{\omega}^+$ with 1s;
while *Termination-Criterion not met* **do**
  foreach $i \in \{1, \ldots, N^{\text{eq}}\}$ **do**
    sample ρ^- according to weights $\boldsymbol{\omega}^-$;
    sample ρ^+ according to weights $\boldsymbol{\omega}^+$;
    $x' \leftarrow \rho^+ \rho^- x$;
    conditional acceptance and update of x, x^* as in SA Alg. 2.1;
    update weights $\boldsymbol{\omega}^-, \boldsymbol{\omega}^+$;
  end
  $T \leftarrow \tau(T)$;
end
return x^*;

In Algorithm 2.2, we present a variant of ALNS in pseudo-code, where the accept/reject mechanism of a proposed neighbor is borrowed from SA. The main differences are the sampling by roulette wheel selection of the operators to create new solutions and the updating of their weights. More concretely, after a corresponding application, the destroy operator pair receives a score ψ, depending on four different outcomes—new best solution, better solution, accepted, rejected. This is used to gradually update the weights via the weight update rule $\omega_\rho \leftarrow \lambda\omega_\rho + (1-\lambda)\psi$, where ω_ρ is operator ρ 's weight and $\lambda \in [0, 1]$ controls the weight decay. The vector of destroy operators are denoted by $\boldsymbol{\omega}^-$ and for repair operators by $\boldsymbol{\omega}^+$.

As hinted in the handbook of metaheuristics [66, Chapter 13], the scores could be normalized by a measure of time for the operators, otherwise, time-consuming operators that automatically lead to better solutions have an unfair advantage, with a potentially negative impact on the diversification of the search. Another important aspect is the size of the fraction of the solution (which can itself be sampled) that is destroyed, which should not be too large, to not degenerate to an iterated greedy approach, and not too small, to not be too constricted in the reconstruction. Again, one sees that many parameters are involved and automated tuning of an ALNS could be beneficial, e.g., using irace [103].

13	10	11	6
5	7	4	8
1	12	14	9
3	15	2	

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

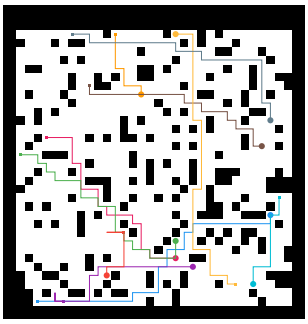


Figure 2.2: Left: Example initial state and goal state of the 15-puzzle. Right: Solution to example MAPF problem instance (planned agents paths), created with Keisuke Okumura's *mapf-visualizer* (`https://github.com/Kei18/mapf-visualizer`).

2.2 State Space Search

In this section, we begin by following Russel and Norvig's famous "Artificial Intelligence: A modern approach" [132, Chapter 3] to introduce search on general artificial intelligence (AI) problems and discuss how this methodology can be transferred to combinatorial optimization problems. There, problem instances are modeled as state graphs, with the goal to find a sequence of actions leading from a given initial state to one of possibly many goal states (if there is one); in the optimization variant a cost-optimal action sequence.

More formally, we are given

- states $s \in \mathcal{S}$, with an initial state s^{I} and a set of goal states $\mathcal{S}^{\mathrm{G}} \subset \mathcal{S}$.
- actions $a \in \mathcal{A}(s)$, potentially depending on the state s,
- a state transition function $\tau(s, a) \mapsto s'$ to move to a new state,
- and costs assigned to such transitions $c(s, a, s')$.

A solution for such an AI problem is a sequence $\mathbf{a} = (a_1, \ldots, a_n)$ of actions transforming the initial to a goal state. A prototypical problem is the 15-puzzle, where on a 4×4 grid, tiles numbered from 1 to 15 have to be sorted by orthogonal moves having one empty field available. A shortest sequence of such moves has to be found for an example initial state as depicted in Figure 2.2 (right). A sequence of actions for another example problem could be jointly planned moves of robots in a time-discrete grid world so that they reach their respective, mutually different goal positions on said grid, without colliding with each other, while minimizing the sum of timesteps over the robots to achieve this—the multi-agent path-finding problem (MAPF, [140]). An example solution of such planned paths is shown in Figure 2.2 (left). To find such a sequence, a graph search algorithm can be employed, a systematic exploration of the underlying state graph until a termination criterion is reached.

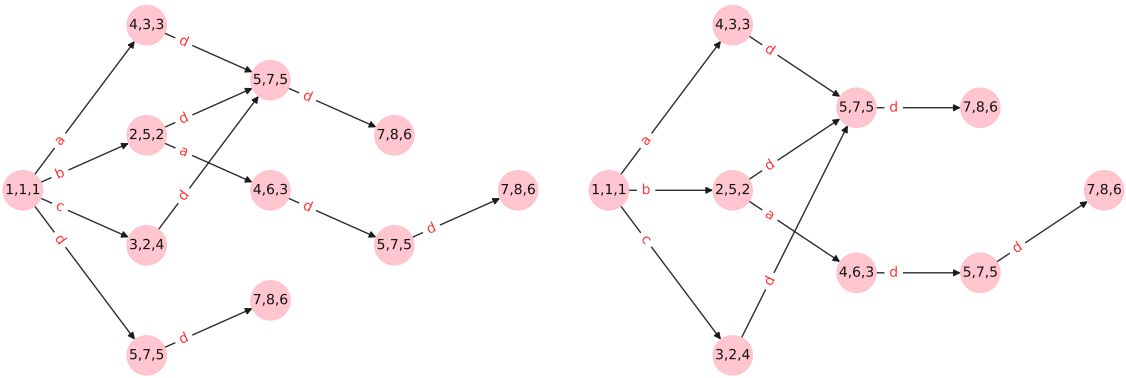


Figure 2.3: Layered state graph for the example LCS problem instance with $S = \{bcadcdc, caabadd, bacddcd\}$ and $\Sigma = \{a, b, c, d\}$ from Blum, Blesa, and Lopez-Ibanez [19]. On the right with filtering of dominated letters, left without. The single longest common subsequence is *badd*.

Combinatorial optimization problems. Splitting a solution to a problem into a sequence of steps is a natural problem solving paradigm and can likewise be applied to combinatorial optimization problems. There, the actions of a *search agent* amount to extending a partial solution like assigning a value to a decision variable or selecting an element to be part of the solution under construction. Goal states are of such kind that they cannot be extended further.

We consider the longest common subsequence (LCS) problem as an example, where we are given m strings $S = (s_1, \ldots, s_m)$ with maximum length n from an alphabet Σ. With a fixed number of strings m, the problem is solvable in polynomial time with increasing n, for m being an input parameter it is an $\mathcal{NP}$-hard problem [106]. Any string t that can be derived from another string s by deletion of characters while keeping the remaining characters in exactly this order is called a subsequence thereof. The goal is now to find a string t^* with maximum length that is a subsequence of all strings s_i. A meaningful state graph is formally defined by Đukanović, Raidl, and Blum [41], where the key elements are:

- Every state v (v for a vertex in the state graph) is induced by a common subsequence t of length l_v, constructed from left to right.
- States are encoded by a position vector where the i-th element is the smallest begin position of the remaining substring of the i-th string, s.t. t is still a subsequence for the string excluding this remaining substring.
- A feasible extension (action) makes a transition from state v to v' by choosing an $a \in \Sigma$, where a occurs at least once in all remaining substrings, increasing the length (costs/reward) by one.

Algorithm 2.3: Algorithmic pattern for state space search.

Input: initial state s^{I}, goal states $\mathcal{S}^{\mathrm{G}}$, action function $\mathcal{A}$, transition function τ, state-action cost function c, heuristic function h, primal cut u
Output: solution encoded sequence of actions $\mathbf{a}$ or *no solution found*

$\mathbf{a}^{\mathrm{best}} \leftarrow (), s^{\mathrm{best}} \leftarrow \textsc{Nil}$;
search frontier $Q \leftarrow \{s^{\mathrm{I}}\}$, reached states $\mathcal{R}$;
while *Termination-Criterion not met* **do**
  $s \leftarrow$ next-state-from(Q);
  if $s \in \mathcal{S}^{\mathrm{G}} \wedge \left(s^{\mathrm{best}} = \textsc{Nil} \vee g(s) < g(s^{\mathrm{best}})\right)$ **then**
    $s^{\mathrm{best}} \leftarrow s$, $\mathbf{a}^{\mathrm{best}} \leftarrow$ derive-solution-from(s^{best});
  end
  for $s' \leftarrow \tau(s,a) | a \in \mathcal{A}(s)$ **do**
    $\tilde{g} \leftarrow g(s) + c(s,a,s'), \tilde{f} \leftarrow \tilde{g} + h(s)$;
    if $\tilde{f} \leq u \wedge (s' \notin \mathcal{R} \vee \tilde{g} < g(s'))$ **then**
      $\mathcal{R} \leftarrow \mathcal{R} \cup s'$, $g(s') \leftarrow \tilde{g}$;
      incorporate(Q, s', h);
    end
  end
end
if $\mathbf{a}^{\mathrm{best}} \neq ()$ **then**
  return $\mathbf{a}^{\mathrm{best}}$;
else
  return *no solution found*;

- Dominated letters $b \in \Sigma$ are not considered, i.e., for which there exists another letter a which results in a better position vector, i.e., there would be no string with a shorter remaining substring when choosing a over b.
- The initial state is induced by the empty string, states without feasible extensions are goal states.

The resulting state graph is a finite directed acyclic graph with a depth bound of n. A longest path from the initial to a goal state corresponds to a longest common subsequence t^*. Example state graphs of an example problem instance of the literature are displayed in Figure 2.3.

Best-first search. When we make use of problem-specific knowledge, e.g., by means of a heuristic function evaluating how desirable a node is for further consideration, we speak of *informed search*, as opposed to uninformed search, cf. Russell and Norvig [132, Chapters 3 and 4]. One well-known informed search method is A* search by Hart, Nilsson,

and Raphael (1968, [78]); it first expands better-ranked nodes according to the so-called f-value which includes the currently best known costs-so-far $g(s)$ and a heuristic $h(s)$, an estimate of the costs-to-go to the nearest goal state from the current state s:

$$f(s) = g(s) + h(s) \tag{2.2}$$

Hence it is a best-first search, as opposed to breadth-first (BFS) and depth-first search (DFS). The basic algorithmic pattern of a state space search is shown in the pseudocode in Algorithm 2.3. As input, the implicit state-space specification as described before is provided and either a solution is returned or the statement that none has been found. We assume minimization of the costs, where maximization can be modeled by negative costs, corresponding to the prize or reward we collect for an action. The main loop selects a next state (or somewhat interchangeably called *search node*) from the current search frontier, checks and conditionally updates the currently best-found solution to a goal node, and expands the successors (children) of the node, extending the search frontier. As standard termination criterion, we stop after having pulled a goal state from the search frontier. If we select always a node with the smallest f-value as the next state from the search frontier, we end up with A* search, if h is non-trivial—if $h(s) \equiv 0$, then this amounts to Dijkstra's classical shortest path algorithm. If we use a queue for the search frontier, we end up with BFS, for a stack with DFS.

Optimal efficiency. We assume that the state graph is finite, which is true by definition for combinatorial optimization problems, and that there is no zero-cost cycle, e.g., every subsequence eventually results in a finite cost increase. Furthermore, let h be *admissible*, i.e., it never overestimates the real costs-to-go h^*:

$$h(s) \leq h^*(s) \; \forall s \in \mathcal{S}. \tag{2.3}$$

A well-known corollary is then that A* is *cost-optimal* and *complete*, i.e., it returns a solution with minimum costs, if there is such, otherwise it terminates with the statement that no solution exists. If the heuristic is *consistent* (also called monotonous), i.e.,

$$h(s) \leq c(s, a, s') + h(s') \; \forall s' = \tau(s, a), a \in \mathcal{A}(s), s \in \mathcal{S}, \tag{2.4}$$

then A* is *optimally efficient*. This means that no cost-optimal path-extending search algorithm (following our algorithmic pattern) with the same heuristic can expand fewer nodes up to tie-breaking for the nodes with $f(s) = C^*$, where C^* are the optimal costs. An intuitive argument is that for cost-optimality we cannot neglect reachable states with $f(s) < C^*$, otherwise, we might miss a cost-optimal path or must have used additional information. Furthermore, for consistent heuristics, states are once expanded not touched again, since the f-values cannot decrease during the search. Otherwise, shorter paths to already expanded states can be found which have to be re-expanded. Cases can then be constructed where A* is not optimal, as discussed in detail by Dechter and Pearl [37].

Space complexity. One major drawback of A* is the worst-case exponential space complexity since all newly visited states have to be retained in the search frontier (open list) until expanded and are then moved to the closed list for solution reconstruction and to check for duplicate states, e.g., due to cycles in the state space or multiple paths reaching the same state. To overcome this, iterative-deepening A* (IDA*) has been devised combing a memory-efficient depth-first search with the optimality results (under some assumptions) of A*. It iteratively runs a DFS with an increasing cut-off value on the costs (called u in Algorithm 2.3), starting from the initial root heuristic estimate and always setting u to the minimum f-value over the cut-off nodes. It terminates when a goal state is found and with an admissible heuristic, the corresponding path is an optimal solution. The DFS is implemented recursively and only has to store the current path of states, leading to a space complexity of $\mathcal{O}(d)$, where d is the maximum search depth.

We assume a consistent heuristic and a most-recent state tie-breaking rule, and that we have tree-like search graphs. Then, in the last iteration, IDA* expands the same set of states as A* without using an open or closed list and the previous iterations do not affect the asymptotic time complexity. For a detailed analysis by Korf with proofs see [93]. In the end, an empirical study is necessary for the concrete problem instances, as to whether A* with its overhead of keeping the states in memory (if there is sufficient), generally allowing duplicate checks at the price of more expensive state expansions potentially reducing the number of expanded nodes, results in an overall faster runtime than IDA*.

Another approach aiming for memory efficiency is frontier search [96], where the closed list (reached list) is either completely abandoned or only nodes close to the frontier are kept. A tricky question is then how to reconstruct solutions. The standard method is to keep in each search node a pointer to the predecessors from which the best path was found to backtrack from a goal state to the initial state. In frontier search, the trick is to apply a divide-and-conquer method. Instead of saving all reached nodes, only those in an intermediate layer are kept (if we have means to estimate the number of actions) and pointers are stored to those. If the search is finished, the problem is split into a search from the start to the intermediate state on the best path, and from this intermediate state to the goal state. This is applied recursively until the problem becomes trivial.

Bounded suboptimal search. Another approach to reduce computational effort and memory footprint is to sacrifice optimality, since it may be satisfactory to find a high-quality solution faster as solving to optimality is impractical anyway. A well-known modification of the A* algorithm is *weighted A** (WA*) introduced by Pohl in 1970 [118], where the heuristic is weighted by a constant factor $1 + \varepsilon \geq 1$:

$$f_\varepsilon(s) = g(s) + (1 + \varepsilon)h(s) \tag{2.5}$$

It follows that the quality of the solution found by WA* C is bounded by $C^* \leq C \leq (1 + \varepsilon)C^*$. For many problems, an empirical observation is that increasing ε results in fewer expansions to find a goal node. But this is not necessarily the case as discussed by Wilt and Ruml [164]. For some problems with many local optima regarding an

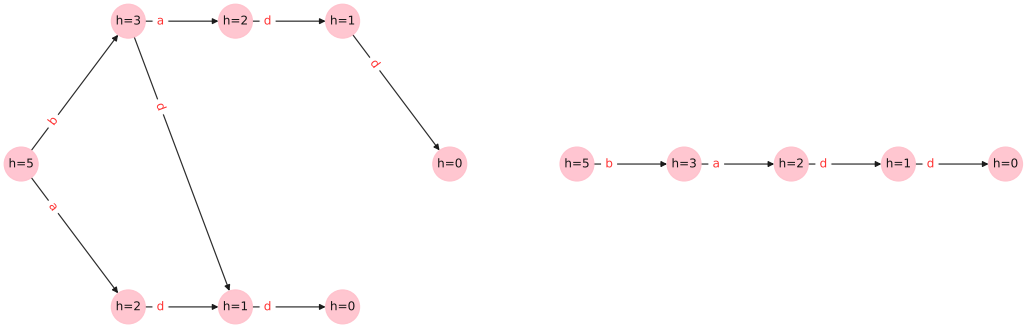


Figure 2.4: Beam search with beam widths two and one (the latter is equivalent to a greedy construction), guided by the upper bound which sums over the alphabet the minimum number of occurrences over the remaining substrings. Both still find the optimal solution since the guidance is almost perfect in this case, except for the root state.

inaccurate h, greedy best-first search ($\varepsilon = \infty$) may get stuck in them and would not find a solution quickly. A^* on the other hand would skip them more quickly since it also considers the costs-so-far g and does not only trust h, so larger values of ε would eventually result in more states being expanded. Due to its astounding simplicity, it is a natural step to study WA*'s behavior when an A* solver is already available.

Beam search. Many variants of beam search exist, with the defining property that the search frontier is of bounded size over the whole search. For instance, we could modify the incorporate function in our algorithm 3.1, as to keep the open list bounded by a beam width β, i.e., if it is full and we need to add another node, it enters the beam if it is better in the f-value than the worst f-value $f_{\max}$ in the beam and a state is at the same time removed from the beam with $f_{\max}$. The most common variant in combinatorial optimization though is a layer-wise truncated breadth-first search. For COPs, the state graph is a directed acyclic graph, so we know that we expand polynomially many states $\mathcal{O}(\beta d)$ in the maximum depth d. In each layer, the successor states are sorted by f and the β best states are kept. If the evaluation function runs in polynomial time, so does the whole beam search.

Beam search has been introduced by Lowerre [105] for a speech recognition system and studied in detail for scheduling by Ow and Morton [113]. Its main drawback is that the basic variant is incomplete, so complete anytime variants have been proposed by Zhang [165] (iterative weakening) and Zhou and Hansen [167] (beam-stack search). Another natural method is iterative broadening/widening [68], where the beam width is iteratively increased and additional pruning is optionally applied, when a dual bound is available, similar to heuristic breadth-first search by Zhou and Hansen [168].

In Figure 2.4 example beam search runs with beam widths two and one (which degenerates to a greedy construction) are shown on the LCS instance from before, where the exact state graph had a width of three (or four without filtering of dominated letters).

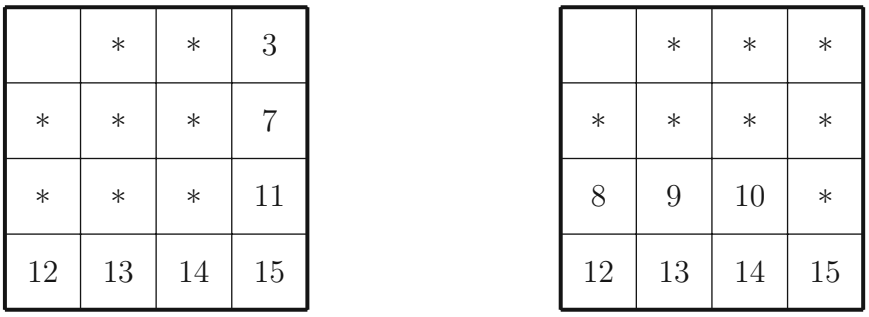


Figure 2.5: Left the fringe pattern, right the corner pattern for the 15-puzzle from Culberson and Schaeffer [32].

Pattern databases. Crucial for the performance of state space search is the quality of the heuristic, which more in the case of optimal solver relates to the pruning capability to exclude states that are not on an optimal path as early as possible and more in the inexact case to the guidance towards more promising nodes by ranking them as similar to the real costs-to-go as possible. One major tool to create strong heuristics are *pattern databases* by Culberson and Schaeffer [32]. The idea is to decompose the problem into subproblems or define subgoals that have to be passed to reach the goal, solve them once beforehand, and save the results in the database, which can then be used to quickly calculate strong heuristics when solving the actual problem.

In Figure 2.5 the subgoal states of two such databases for the 15-puzzle are shown, where seven tiles and the blank have to be brought into their final position, while the other tiles remain abstract but their moves are counted. The shortest path length from any state to that pattern is calculated and stored in the eponymous database, which itself is an admissible heuristic. During the search, for the calculation of h, indices for the given databases are calculated and the maximum over the retrieved values is taken, which itself is admissible. Together with symmetry considerations, this allowed to reduce the number of expanded nodes for an IDA* search to be reduced by a factor of 1 000 [32].

Korf and Felner [94] in 2002 pushed this idea further and suggested a disjoint decomposition into subproblems, where elements of the states (e.g., the tiles in the 15-puzzle) are decomposed into disjoint groups and only moves within the groups count towards the solution length. The calculation of an admissible heuristic can then also be achieved by summation instead of only taking the maximum, resulting in stronger heuristics. They were able to practically solve random 24-puzzle instances within days using this approach, before only achievable with a more complicated heuristic and a finite state machine (FSM) based duplicate state pruning technique [95]. Felner et al. [50] elaborate on the general idea of additivity further and present a detailed study of additive pattern database heuristics on a wide set of planning and optimization problems.

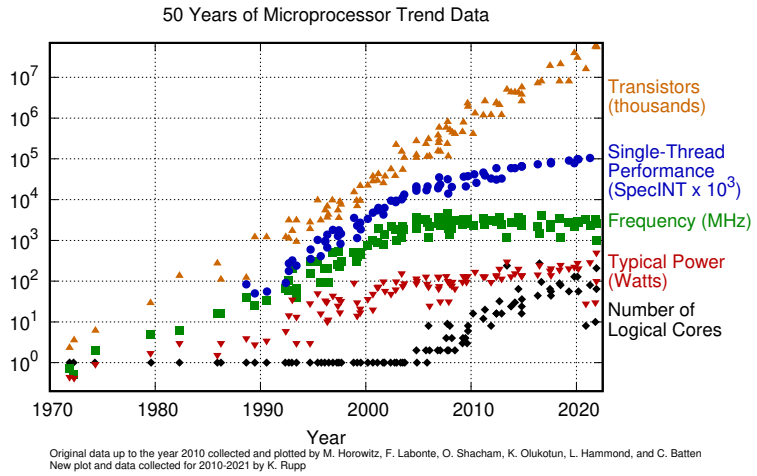


Figure 2.6: From `https://github.com/karlrupp/microprocessor-trend-data` by Karl Rupp, licensed under `https://creativecommons.org/licenses/by/4.0/`. In the last decade, the single-threaded performance increase has declined and application performance increase more relies on parallelization.

2.3 Shared-Memory High Performance Computing

Solving optimization problems well is intricately connected to efficient and fast implementations on modern computer architectures, exploiting their capabilities while avoiding pitfalls and being aware of performance bottlenecks. In the last two decades, the rate of increase of single-threaded performance has declined and for an overall application performance increase focus on parallelization on multicore architectures has become even more important (see Figure 2.6). In this section, we briefly discuss selected crucial aspects for high performance computing (HPC) on cache-based multicore processors based on the first five chapters of the practice-oriented book "Introduction to High Performance Computing for Scientists and Engineers" by Hager and Wellein [77]. For implementations, we follow the practices for HPC for the LLVM JIT-compiled language Julia described in "Julia High Performance - Second Edition" by Sengupta and Edelman [134], which is our HPC language of choice, also with increasing popularity in the scientific community.

Uniform vs. non-uniform memory access. In this work, we focus on parallelization on multi-core systems with shared memory. Multiple processors or cores have direct, transparent access to the main memory via a bus or crossbar switches, see Figure 2.7. If the behavior over the whole physical memory is the same—uniform—over the whole address space, we speak of uniform memory access (UMA). A common performance pitfall is to neglect so-called non-uniform memory access (NUMA, Figure 2.7 on the right), where processors/cores are grouped into NUMA domains with fast access to their local memory but relatively slow to other NUMA domains.

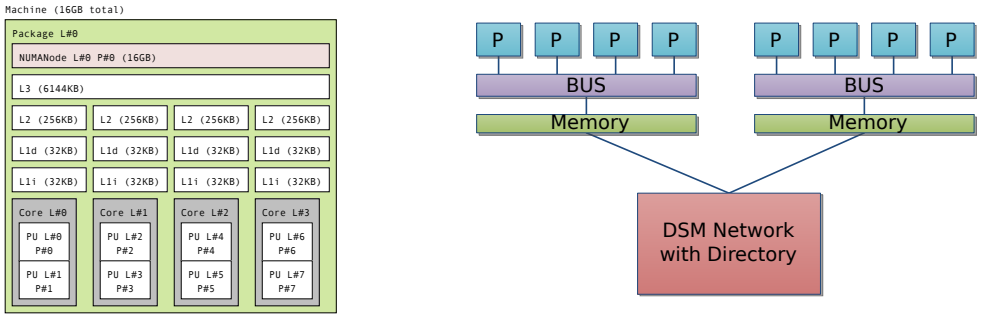


Figure 2.7: Left: Visualization of a shared-memory architecture with one NUMA domain on a Macbook Pro Mid 2015 with Intel Core i7 processors created with OpenMPI's hwloc/lstopo [25] (`https://www-lb.open-mpi.org/projects/hwloc/`). Right: NUMA with multiple domains and processor groups, each with their own local memory, visualization from `https://commons.wikimedia.org/wiki/File:NUMA.svg`

Pipelining and caching. Modern computer architectures suffer from the "DRAM gap" or the "memory wall". The speed increase of the memory is below the one for microprocessors, the two have diverged. In particular, it has a rather high latency on random access (time until data becomes available on the bus) and limited bandwidth. Therefore, a key challenge of fast computing is that the arithmetic (integer/floating point) logical units (ALUs) that execute the desired instructions are fed at a sufficient rate with data. Instructions are retrieved from the main memory, interpreted, and executed involving the necessary units of a CPU core. In scientific computing, we often encounter a streaming pattern, where sequential bulks of data are retrieved from memory, arithmetically and logically manipulated, and stored back to memory. Two main concepts enable good streaming behavior, namely pipelining and caching.

In pipelining, operations that concern different units of the CPU are split into instructions that are executed staggered in parallel operating on different data. The output of each stage's instruction is the input of the subsequent stage. In particular loading data to registers and storing to memory would stall the arithmetical unit, if not performed in parallel. Pipelining allows increasing the average number of results delivered (e.g., floating point operations) per second, depending on the depth of a pipeline (i.e., the wind-up where not all workers are busy yet and wind-down where not all workers are busy anymore) and the number of performed iterations (e.g., length of the vector on which a pipelined manipulation is performed).

In Figure 2.8, we see an example pipeline of four instructions where each consists of four operations (fetch, decode, execute, write-back). Executing this set alone sequentially would take 16 cycles, due to the pipeline and independent micro-instructions, it only takes 7 cycles (in cycle number 8 we are done). The throughput is the number of independent instructions N divided by the steps needed to execute them $N + m - 1$ in a depth-m pipeline and calculates to $(1 + (m - 1)/N)^{-1}$. The shorter the m and the larger the N, the closer we are to the ideal (neglecting superscalarity) throughput of one instruction per cycle.

To bridge the memory gap, caching hierarchies have been employed, which serve as

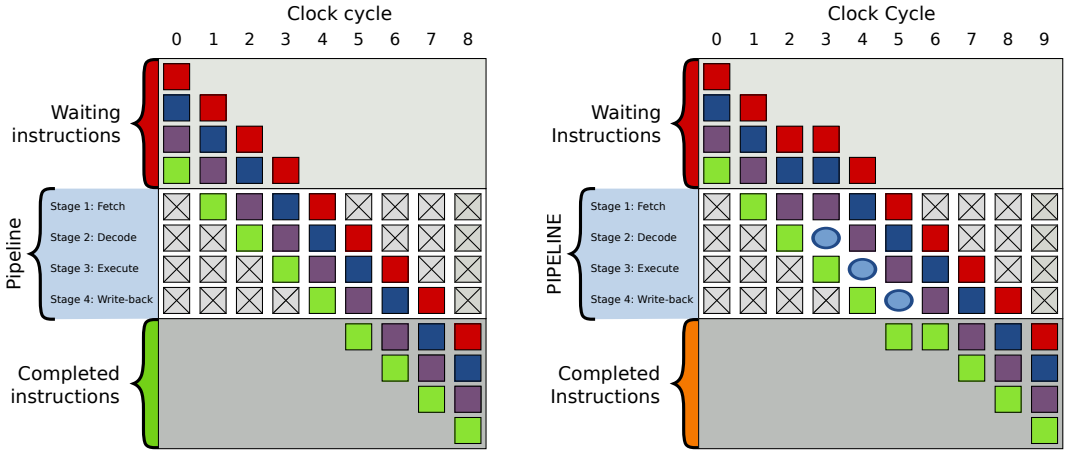


Figure 2.8: Example pipeline of depth four where each instruction takes four clock cycles, from `https://en.wikipedia.org/wiki/User:Cburnett` licensed under `https://creativecommons.org/licenses/by-sa/3.0/`. Right with a bubble, where an instruction is stalled due to missing data.

intermediate, high-speed stores for instructions (L1i) and data (L1d, L2, L3). The closer to the core (L1), the faster but the smaller. Instead of single-word interactions with memory, they are retrieved and saved in so-called *cache lines*. The assumption is that applications obey to some extent a spatial (data close to already retrieved one is more likely to be retrieved sooner than other) and temporal (already retrieved data is likely to be reused again soon) access pattern. This is why it makes sense to store data in the cache where it can be quickly loaded from and stored in the registers. Furthermore, prefetching is employed to load instructions or data from higher cache levels or the main memory when it is likely to be needed soon, i.e., the chunk of instructions or bytes of a vector.

In Figure 2.9, we see a combination of both effects when considering a simple stream manipulation of three length-N vectors $\mathbf{B}, \mathbf{C}, \mathbf{D}$ of reals. We calculate the vector triad $\mathbf{B} + \mathbf{C} \cdot \mathbf{D}$ by a for loop that iterates over all elements and stores the results in a vector $\mathbf{A}$, each denoting two floating point operations (FLOPS), multiplication and addition. Special SIMD instructions are not employed, which perform multiple instructions in parallel on a sequence of data like four floats in parallel. We recreated this example from [77, Chapter 1] and tested it on two different platforms (AMD EPYC, Intel Core) with different clock frequencies and cache layouts, once with single and once with double precision. We observe that in the beginning there is a somewhat steady increase in the MFLOPS/sec, which can be largely attributed to the increasing pipelining throughput and that the whole vectors can be streamed from the L1 cache. Starting from roughly 10^3 there is a sudden drop in performance. This is when the L1 cache (in the range of 32 to 64 KB per core) cannot hold the whole vectors anymore and has to perform costly interactions with L2 cache (more cache misses and write-backs occur, also causing pipeline bubbles, see Figure 2.8, where data is not ready at a certain stage). Further drops occur when L2 is not large enough anymore and either L3 cache has to be used, or

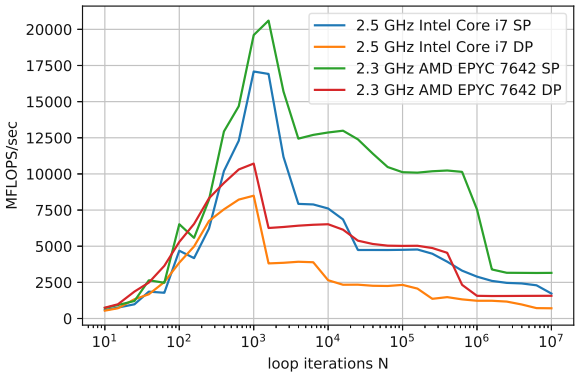


Figure 2.9: Recreated vector triad operation example from [77, Chapter 1] in Julia 1.8 for single precision (SP) and double precision (DP) floats.

eventually the data has to be streamed directly from and to memory. The exact analysis is much more intricate and also depends on compiler optimizations but should convince us that cache-friendly implementations are important.

Data parallelism, cache coherence, and false sharing. A common parallelization pattern is to split the input data across multiple workers and combine the results into a single output in a final synchronization step for further processing. The operations applied to the data elements are the same and independent from one another. This is well-suited when the sequential overhead to distribute the workload and join the results is relatively small. For instance, we could evaluate a set of nodes in a layer of a search tree or calculate a histogram over a list of numbers by data parallelism.

A common pitfall and anti-pattern related to data parallelism on shared-memory systems is *false sharing*. This happens when the cache lines related to the seemingly independent memory regions assigned to each thread overlap causing undesired reads and premature write-backs to the main memory to ensure cache coherence. This can have a severe impact on cache efficiency which is crucial for high performance on cache-based architectures as discussed before, as memory access is orders of magnitude slower than cache access.

As an example, we consider the task of calculating a histogram for a given input array of N numbers into b bins. In a data-parallel algorithm (visualized in Figure 2.10), we first create a $b \times \tau$ matrix, where each of the τ threads has its own histogram bins. A static scheduler then assigns τ consecutive approximately equal chunks to the threads which then loop over the data to calculate the bin of each element and increase a counter of their zero-initialized thread-local histograms. In a final sequential synchronization step, the matrix is reduced to a vector of length b by summing over the corresponding axis, encoding the histogram, our final result.

We implement this in Julia and measure the runtime over the number of threads on an AMD EPYC 7642 with uniform memory access for $N = 1.5 \cdot 10^9$. The memory layout of the $b \times \tau$ matrix is actually a linear vector of length $b \cdot \tau$. A common pitfall here is

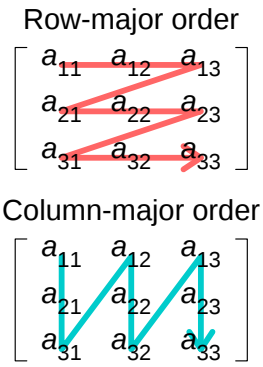


Figure 2.11: Left: parallel histogram creation speedups, once with false sharing where we observe an irregular speedup behavior due to strong cache coherence related thread interference. Right: row-major vs. column-major order, created `https://commons.wikimedia.org/wiki/User:Cmglee` licensed under `https://creativecommons.org/licenses/by-sa/4.0/`.

the distinction between row-major order, where columns are neighbors, as used in C or Python, and column-major order, where rows are neighbors, as in Fortran or Julia, see Figure 2.11 on the right. In Julia we, therefore, need to create a matrix with b rows to have the histogram bins contiguous in memory for access locality and τ columns, to separate the thread-local regions from one another, adding a cache line length to b as a safety buffer. When we do it the opposite way we end up with false sharing and strided memory access. On the left of Figure 2.11 we see the runtime of the number of threads and the irregular speedup behavior with false sharing, which is even below one in the beginning and approaches three for 46 threads. Without sharing, the shape looks much better and we end up with a speedup of over 13. This is also not too strong but likely due to the small amount of work in the loop body and the task being more memory-bound.

Load balancing. Keeping the threads in parallel sections busy as much as possible is another important aspect to achieve high parallel efficiency. In the histogram example from before, the approximate splitting in equal chunks of data assigned to threads was sufficient, since the executed instructions were data-independent and we resolved false

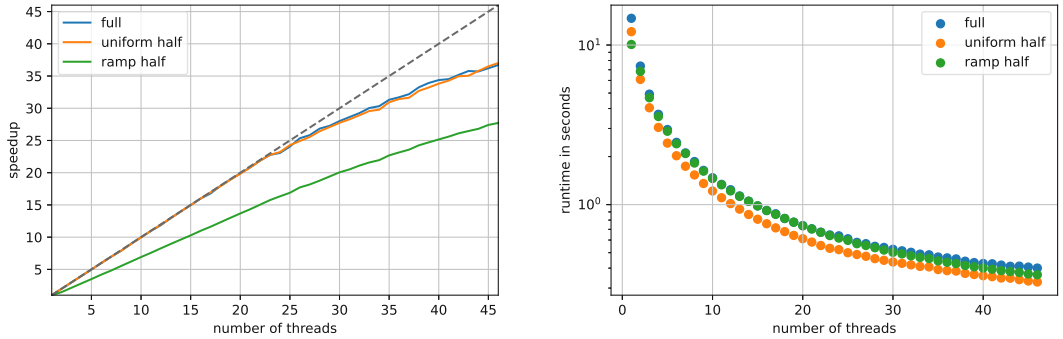


Figure 2.12: Comparison of speedups and runtimes for different load patterns, where for active numbers the sin $\sqrt{x}$ has to be calculated.

sharing to avoid interference between threads. For more complex tasks, as we will see in our state-space search settings, the performed instructions and actual runtime may well vary due to data-dependent branching and loops. If the load is not well-balanced over the threads, the longest-running thread up to a synchronization point may then substantially reduce parallel efficiency and limit speedup.

If the number of elements is large and there is no hidden structure, we can assume in a first step that this variance is smoothed out in the average over many elements processed by each thread with only a minor impact efficiency. We consider another simple example to study the principle effects. We are given again a list of N random numbers sampled uniformly from $[0, 1]$, while this time we perform a computationally more demanding task calculating for each number x the sine of the square root $\sin \sqrt{x}$. Additionally, a boolean masking vector is provided that states whether a calculation for a number should be performed or not, separating active from inactive data.

We compare three different masks, one where all numbers are active (full), one with an equal probability of 0.5 over all numbers (uniform half), and one with a skewed distribution where the probability for a number to be active decreases linearly with the index, starting from 1.0 for the first to 0.0 for the last index (ramp half). We implement this again in Julia and measure the speedup over the number of threads on an AMD EPYC 7642 processor with uniform memory access for up to $N = 1 \cdot 10^9$ numbers. In Figure 2.12 we show corresponding speedups and runtimes. For the first two load patterns, we observe a nearly perfect speedup up to 23 threads and then a gradual decline until a speedup of 36× for 46 threads, a parallel efficiency of approximately 80%. The ramp on the other hand suffers from workload imbalance since the threads for memory regions with smaller indices have a higher average number of calculations to perform. Uniform half and ramp have almost the same number of actual calculations to perform ($N/2$), but the single-threaded runtime for ramp is smaller, which is likely due to the benefit of more access locality/cache efficiency and more successful branch prediction.

2.4 Modeling and Solving Dynamic Problems

So far, we have considered perfectly observable, deterministic problems. In real-world problems, we frequently have to consider uncertainties regarding the perception of our surroundings, their evolution, and the effect of actions an agent performs. A common way to model these are Markov decision processes (MDPs, [122]), where we are again given states and actions, but the transitions between the states and potentially also the reward (replacing the term costs) are probabilistic. Still, we assume the Markov property, where a state $s \in \mathcal{S}$ contains all the information we need to make statements about the future, i.e., we do not need to know the history of state-actions transitions to arrive at the current state.

In this section, we briefly describe the main elements of MDPs relevant for us, where we follow the treatments of Sutton and Barto [142, Chapter 3] in the context of reinforcement learning and Powell [119, Chapters 2,3,5] in the more suitable context of approximate dynamic programming (ADP), focusing on practical aspects and less the underlying theory. We later use it in the main body to model dynamic and stochastic vehicle routing problems as done in the literature, e.g., by Voccia et al. [162], Ulmer et al. [149, 150].

Basic modeling framework. Following Powell [119, Chapters 2], there are some basic ingredients to formulate a stochastic and dynamic optimization problem with a finite time horizon:

- discrete time steps $t \in \{0, \dots, T\}$, also called decision epochs,
- states S_t, containing all the information we need to make an informed decision,
- state and therefore time-dependent actions $a_t \in A(S_t)$,
- an exogenous process W_t, e.g., a process with known probability distribution that creates orders to be served,
- a transition function $S_{t+1} = S^M(S_t, a_t, W_{t+1})$,
- and a contribution $C_t(S_t, a_t)$, the costs/reward when applying action a_t while in state S_t.

The goal is to find a policy $\pi \in \Pi$—a decision rule $A^\pi(S_t)$ selecting the action in a state—to maximize the expected contribution (also called reward):

$$\max_{\pi \in \Pi} \mathbb{E} \sum_{t=0}^{T} C_t(S_t, A^\pi(S_t)) \tag{2.6}$$

Small problems, for which both state and action space are limited, can be solved recursively by backward dynamic programming to calculate the value of each state

$$V_t(S_t) = \max_{a_t} \left(C_t(S_t, a_t) + \mathbb{E}\{V_{t+1}(S_{t+1}) | S_t\} \right) \tag{2.7}$$

Algorithm 2.4: ADP approach for post-decision states using value function approximation, adapted from [119, p. 241].

Input: initial estimate of post-decision value function $\tilde{V}^{a,0}$, number of training iterations N
Output: approximate value function $\tilde{V}^a$

foreach $i \in \{1, \ldots, N\}$ **do**
  $\Omega \leftarrow$ create sample path;
  $S_0 \leftarrow$ initialize start state;
  foreach $(\omega, t) \in \Omega$ **do**
    solve $a^* = \arg\max_{a_t} \left(C_t(S_t, a_t) + \tilde{V}_t^{a,i-1}(S_t^a)\right) | S_{t-1}^a$;
    optionally update from $\tilde{V}^{a,i-1}$ to $\tilde{V}^{a,i}$;
    apply optimal action to move to post-decision state $S_t^a \leftarrow a_t^*(S_t)$;
    move to pre-decision state $S_{t+1} \leftarrow S^M(S_t, a_t, W_{t+1}(\omega))$
  end
  update value function $\tilde{V}^{a,i-1}$ to $\tilde{V}^{a,i}$ using encountered state-value pairs;
end
return $\tilde{V}$

and then select in each state the action that maximizes $C_t(S_t, a_t) + \mathbb{E}\{V_{t+1}(S_{t+1})|S_t\}$.

Approximate dynamic programming. For larger optimization problems, where either the state space and/or the action space grows exponentially with the instance size, backward induction becomes quickly intractable. Instead, we have to apply forward dynamic programming, where we iteratively improve an estimate $\tilde{V}$ of our value function and use it to make decisions by taking the action:

$$\tilde{a} = \arg\max_{a_t} \left(C_t(S_t, a_t) + \mathbb{E}\{\tilde{V}_{t+1}(S_{t+1})|S_t\}\right) \tag{2.8}$$

If we have means to simulate our exogenous process, we can apply what is called Monte Carlo sampling to improve the knowledge about our value function. Introduced by Powell [119], a powerful modeling trick is to separate a state into the pre-decision state, where we have new information but not yet made a decision, and the post-decision state, where we have made a decision but not yet received new information. We call the post-decision state S_t^a to indicate that an action a has been performed. The optimality equation to obtain the value of a post-decision state then looks like:

$$V_{t-1}^a(S_{t-1}^a) = \mathbb{E}\left[\max_{a_t} \left(C_t(S_t, a_t) + V_t^a(S_t^a)\right) \middle| S_{t-1}^a\right], \tag{2.9}$$

The maximization is now inside the expectation. When we are in the post-decision state S_{t-1}^a, we make a transition to the pre-decision state S_t by the next element of

a given sample path. Then, we have a deterministic optimization problem to solve, i.e., maximizing the sum of the contribution of an action in S_t plus the value of the corresponding post-decision state. The expectation is now directly encoded in the value function.

In Algorithm 2.4, an abstract algorithm to improve an initial estimate of a value function $\tilde{V}$ is given. In each iteration, a sample path for the exogenous process is followed, where the optimal decision is used to traverse the state space, according to the sum of the current contribution and the current approximate value of the resulting post-decision state. Furthermore, an update for the value function estimate is performed based on the experience so far in form of state-value pairs. Which exactly is left unspecified here and depends on the concrete algorithm, e.g., it could be an approximate value iteration scheme, where we have a lookup table with values for each state and an α-update rule is applied. Common in reinforcement learning is to employ temporal difference learning, where often a neural network is used to approximate the value. The experience—the replay buffer—is randomly sampled and used to update the weights of the neural networks to reduce the loss on this sample by a stochastic gradient descent step. Another important aspect is to not always select the currently optimal decision based on the—potentially inaccurate—*value function approximation* (VFA), but to also sample random decisions occasionally to explore the state space. More details can be found, e.g., in Sutton and Barto [142, Chapter 9].

A recent methodological survey for stochastic optimization is provided by Powell [120]. Besides VFA, further important methods are *policy function approximation* (PFA), where parameterized functions take the state as input and directly output a decision. Another approximation method, in particular, relevant for us is *cost function approximation* (CFA), where a decision is derived by solving an optimization problem over a suitably modified/augmented contribution function, which we also call surrogate function [97]).

2.5 Supervised Learning for Regression Problems

The goal of supervised learning is to form an agent able to solve a given task with a desired performance making use of labeled training data. For regression problems, the agent's task is to predict or estimate an output depending on input data, so-called features. To achieve this, a set of known input-output examples is available to learn a connection between those. A classical regression problem motivated by physics and astronomy dates back to the 19th century when Legendre and Gauss derived the method of least squares. The setting is an experiment where a series of noisy measurements is performed. The collected data is afterwards fit (using a learning algorithm, often formulated as a continuous optimization problem) to hypotheses, e.g., a family of functions with learnable parameters capturing the suspected relations between input and output. The result is to see how well the data matches the theory and to estimate physical properties of objects or even fundamental constants.

Supervised learning for regression (and classification) has a well-understood theoretical

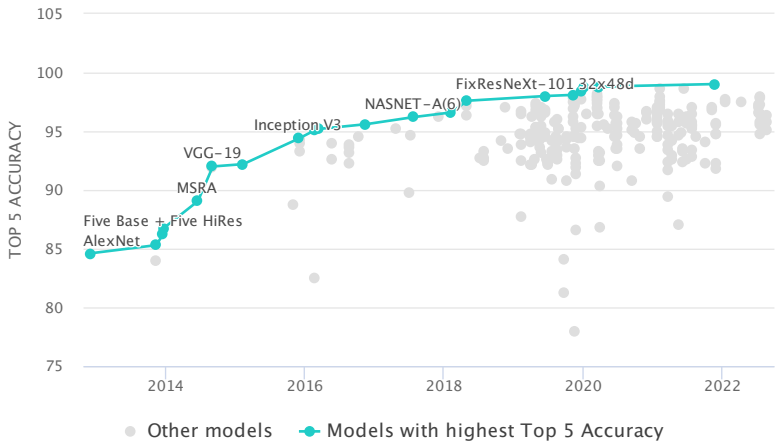


Figure 2.13: Top-5 accuracies appraching more than 99% over time for different network architectures on ImageNet [38] classification challenge, starting with the famous AlexNet [98] based on deep convolutional neural networks, taken from `https://paperswithcode.com/sota/image-classification-on-imagenet` licensed under CC-BY-SA `https://creativecommons.org/licenses/by-sa/4.0/`.

foundation, statistical learning theory, as presented, e.g., by Vapnik [161] and Hastie et al. [79]. We mainly follow the treatment of Goodfellow et al. [75], the "Deep Learning Book"[3] from 2016, in particular Chapters 5 to 7 discussing the basics, with strong focus on neural network based learning for different kinds of problems. Successful applications were made possible by the recent advances in computational power due to massive multicore parallelization by GPUs, availability of huge labeled training data sets, and network architectural and learning algorithm advances. This has led to an AI revolution, in particular in image classification, in the last decade. In Figure 2.13, the increase in top-5 accuracy for the ImageNet image classification challenge consisting of 15 million high-res labeled images categorized in 22 000 classes is shown, starting from the landmark work by Krizhevsky et al. [98], using a deep convolutional neural network (CNN) with 60 million trainable parameters as a classifier.

Model-based approach. Informally, we can state our problem as learning a function $\hat{f}$ that approximates an unknown function f, the *ground truth*, sufficiently well

$$\hat{f}(\boldsymbol{x};\boldsymbol{\theta}) \approx f(\boldsymbol{x}), \tag{2.10}$$

where we are given as input (to our learning problem) a set of m examples $\mathcal{T} = \{(\boldsymbol{x}_i, f(\boldsymbol{x}_i))\}_{i=1}^{m}$—feature/label pairs—which we call the *training set*. We call $\hat{f}$ a *model* with learnable parameters $\boldsymbol{\theta}$. A starting point is often a linear model $\hat{f}(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\beta}^T\boldsymbol{x} + b$, where we assume that the output can be explained by a linear combination of the inputs plus a bias.

The labels are often referred to as $\boldsymbol{y}$ and may themselves be subject to noise, so it could very well be that the same or very similar input features result in a substantially different

[3] `http://www.deeplearningbook.org`

label. It is often implicit that we assume an additive noise model and data is distributed by the random process $y = f(\boldsymbol{x}) + \epsilon(\boldsymbol{x})$, where f is deterministic and ϵ is stochastic. Filtering this noise is precisely the goal of classical regression approaches like the method of least squares. Noise can occur due to limitations of our measurement apparatus—a lack or inaccessibility of information and features—or intrinsic, i.e., based on physical events believed to be truly random.

What we then have is a conditional probability distribution $p(y|\boldsymbol{x})$, i.e., given input $\boldsymbol{x}$, what is the probability of measuring output y. A well-established paradigm in frequentist statistical inference is *maximum likelihood estimation* (MLE). Given a set of measurements $\mathcal{T}$ and a parameterized model distribution $\hat{p}(y|\boldsymbol{x};\boldsymbol{\theta})$, we try to find the parameters $\boldsymbol{\theta}^*$ which maximize the probability of the given measurement $\prod_i \hat{p}(y_i|\boldsymbol{x}_i;\boldsymbol{\theta})$, or equivalently minimizing the negative log-likelihood sum:

$$J_{\mathcal{T}}(\theta) = -\sum_{i=1}^{m} \log \hat{p}(y_i|\boldsymbol{x}_i;\boldsymbol{\theta}) \tag{2.11}$$

Remark: A different philosophy is Bayesian statistics [64], where the model parameters are assumed to have a distribution $p(\boldsymbol{\theta})$ themselves, in contrast to being unknown but fixed. We can introduce prior information about $\boldsymbol{\theta}$ into our learning process and Bayesian statistics provide the machinery to update our current belief over them when new information arrives. Still, if desirable the knowledge about $\boldsymbol{\theta}$ can also be reduced into a single point estimate by the *maximum a-posteriori* (MAP) estimate.

Linear models. As mentioned before, in a first-order approximation of dependencies between $\mathbf{x}$ and y, we often assume a linear model with weights $\boldsymbol{\beta}$ and a bias b, forming together the weights $\boldsymbol{\theta}$:

$$\hat{f}(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\beta}^T\boldsymbol{x} + b \tag{2.12}$$

For instance, we are given a two-dimensional input vector $\mathbf{x} = (x_1, x_2)$ and end up with the model $\hat{f}(\boldsymbol{x};\boldsymbol{\theta}) = \beta_1 x_1 + \beta_2 x_2 + b$. A common trick to capture non-linear relations is the so-called *kernel method*. The input feature vector $\mathbf{x}$ is extended by further elements $\phi_j(\mathbf{x})$, where the ϕ_j are different non-linear transformations. To continue our example from before, assume we add the quadratic relations x_1x_2, x_1^2, and x_2^2. The advantage is that by this kind of transformation, we can apply learning algorithms designed for linear models during which the features are constants. The disadvantage is that we increase the dimensionality of the model and may have to apply regularization to avoid overfitting, as discussed in the next paragraphs. Besides linear models, which we will frequently use in this thesis as a baseline, we briefly discuss feed-forward neural networks at the end of this chapter. They are layered computational graphs also including non-linear functions and have a high capacity to predict arbitrary relations between input and output.

Generalization. The goal of a learning algorithm is to minimize a measure of error of a predictor, called the *loss function*. If we assume the noise to be Gaussian with

zero-mean and constant variance, MLE provides as a measure the mean squared error (MSE):

$$\mathrm{MSE}_{\boldsymbol{\theta}}[\hat{f}] = \mathbb{E}[(y - \hat{f}(\boldsymbol{x}; \boldsymbol{\theta}))^2] \tag{2.13}$$

For practical calculations, this measure is estimated by sample averages, where we assume that the $(\boldsymbol{x}, y)$ follow a data generation distribution $p(\boldsymbol{x}, y)$. We use one sample for training, the aforementioned training set, and one for estimating its performance on unseen data, the test set:

$$\mathrm{MSE}^{\mathrm{data}}_{\boldsymbol{\theta}}[\hat{f}] = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{f}(\boldsymbol{x}_i; \boldsymbol{\theta}))^2 \tag{2.14}$$

If all samples are i.i.d. (identically independently distributed), we can establish a relation between $\mathrm{MSE}^{\mathrm{train}}$ and $\mathrm{MSE}^{\mathrm{test}}$. The test MSE describes the performance on unseen data and must not be used in the training process, otherwise, information might be included in the learned model which is actually not available—it can only use the training data.

This poses the problem of how to make use of the training data to generalize well to the unseen data, which is related to the concepts of model capacity, underfitting, overfitting, and the bias-variance dilemma. Assuming a fixed unseen $\boldsymbol{x}$ with repeated sampling of training data determining different $\hat{f}$, bias is the expected deviation of our prediction from the true value. Variance is the expected squared deviation of our prediction from the expected prediction. If we have some prior knowledge about $p(y|\boldsymbol{x})$, e.g., we assume a linear model, then we introduce a high bias but are robust towards noise, i.e., have low variance. If we have non-linear effects included in our measurements then this may lead to underfitting, our bias is too high. Naturally, we want both to be low but since they are counteracting, we have to find a sweet spot.

Stochastic gradient descent. A well-known learning framework is *stochastic gradient descent* (SGD) or *minibatch gradient descent*. There, we iteratively take a single example or a batch of examples drawn from the training set and calculate the gradient of the loss function $\nabla_{\boldsymbol{\theta}} J$, where J corresponds, e.g., to the MSE. Then we make a step scaled with a learning rate α in the negative direction as the goal to minimize the loss:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J \tag{2.15}$$

The gradient is noisy itself since every time it depends on a different example or batch. An assumption is that we have access to the gradient of our model $\nabla_{\boldsymbol{\theta}} \hat{f}$.

The advantage of SGD and its minibatch variant is the computational efficiency since we do not need to iterate over the whole training batch for each iteration. The outcome of learning algorithms on concrete data depends on so-called hyperparameters (to distinguish them from the learnable parameters), the algorithmic parameters. Common hyperparameters are the weight initialization strategy, model complexity and regularization parameters, the learning rate, and the number of iterations.

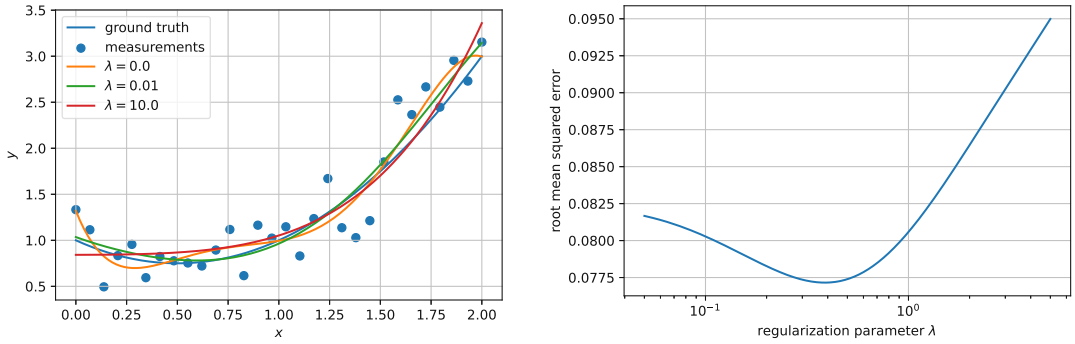


Figure 2.14: Left: Ridge regression examples to fit a fifth-order polynomial to noisy data from ground truth $f(x) = 1 - x + x^2$. Right: Root mean squared error w.r.t. ground truth over increasing regularization parameter λ.

There are many different, more sophisticated variants of gradient-based optimizers [129] where the goal is to improve convergence speed and make them more robust towards noise, e.g., the Adam optimizer [90].

Regularization. It is desirable that our learning algorithm automatically finds the sweet spot for our bias-variance tradeoff. A general concept to avoid overfitting and improve generalization of models is *regularization*. One method is to augment the loss function as to penalize model complexity so that the learning algorithm prefers models with less complexity, but still has sufficient capacity to potentially model more complex relations. A well-known variant is Tikhonov regularization, also called ridge regression or weight decay, where the squared L^2 norm of the weight vector is added to the loss function, resulting in an adapted SGD update rule:

$$J^{\text{reg}}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \alpha||\boldsymbol{\theta}||_2^2 \tag{2.16}$$

$$\boldsymbol{\theta} \leftarrow (1 - \alpha\lambda)\boldsymbol{\theta} - \alpha\nabla_{\boldsymbol{\theta}} J \tag{2.17}$$

At every iteration, the weights are shrunk, controlled by the λ hyperparameter.

In Figure 2.14, ridge regression fits using scikit-learn [115] of a fifth-order polynomial to noisy (zero-mean Gaussian with $\sigma = 0.25$) data over different λ are shown, to see the transition between overfitting ($\lambda = 0.0$) and underfitting ($\lambda = 10$), loosely corresponding to a capacity transition of the model.

Feed-forward neural networks. In their feed-forward variant, artificial neural networks (ANNs), also called multilayer feed-forward perceptrons, are computational graphs with an input $\boldsymbol{x}$, an output y, and $k+1$ layers ($k-1$ hidden layers, one input layer, and one output layer). Their building blocks are neurons (called units) i in layer j which perform an affine transformation of their inputs defined by the weights W_{ji} and biases b_{ji}, apply an activation function z_{ji}, and pass the resulting signal on to their output, potentially to all the next layer's neurons in the fully connected case. The behavior of a neural network based model $\hat{f}$ is determined by its architecture, the edge weight matrices

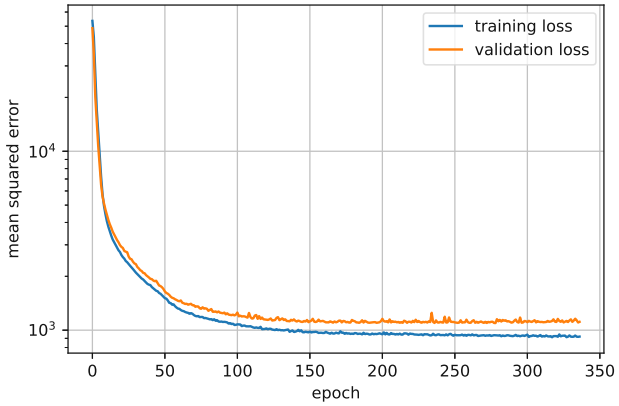


Figure 2.15: Example learning curve using TensorFlow/Keras and validating a neural network with Adam optimizer on a regression task.

$\{\mathbf{W}_j\}_{j\in 1,\dots,k}$ and biases $\{\mathbf{b}_j\}_{j\in 1,\dots,k}$—the learnable parameters $\boldsymbol{\theta}$—and the activation functions $\{\mathbf{z}_j\}_{j\in 1,\dots,k}$ over all layers, where for regression the final activation function z_k is scalar, i.e., we have one neuron in the output layer:

$$\hat{f}(\boldsymbol{x};\boldsymbol{\theta}) = z_k(\mathbf{W}_k^T \dots \mathbf{z}_2(\mathbf{W}_2^T \mathbf{z}_1(\mathbf{W}_1^T \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) + \dots + \mathbf{b}_k) \tag{2.18}$$

By the famous results of Cybenko [33], Hornik et al. [83], and later Leshno et al. [100], with at least one hidden layer, sufficiently many units, and non-polynomial activations, a neural network acts as a universal function approximator. It is also trainable by gradient-based methods using the famous backpropagation algorithm by Rumelhart et al. [130]. This makes them an ideal starting point for black-box modeling, where we are not so much interested in the concrete inner workings of our model but merely in its behavior in terms of input-output relations.

Due to its potential high capacity overfitting is a serious issue. Hyperparameter tuning and regularization are even more important than with high-bias models. While weight decay is again an important basic technique, further techniques have been developed specifically for neural networks like dropout (Srivastava et al. [139]), batch normalization (Ioffe and Szegedy [87]), and—in particular for the setting of sparse data for image classification tasks—data augmentation (Shorten and Khoshgoftaar [138]). A simple yet effective technique is early stopping, where during an iterative training procedure, the performance on an independent validation set is monitored (see Figure 2.15) and stopped as soon as there is no more improvement after a number of iterations. Then, we revert to the weights of the best-performing iteration.

CHAPTER 3

State Space Search for the Traveling Tournament Problem

The traveling tournament problem (TTP) introduced by Easton, Nemhauser, and Trick [46] in 2001 is a well-known sports league scheduling problem famous for its practical hardness. Given an even number of teams with symmetric distances between their venues, a double round robin tournament has to be scheduled minimizing the total travel distances over all teams. We consider the most common constrained variant disallowing games back-to-back between two teams and setting a limit on consecutive home/away games of three. At the time of writing, only instances with as little as 10 teams have been solved to optimality using a parallel iterative deepening A* search [155], while those with 12 teams remain open.

Inspired by the state-of-the-art exact tree search approaches by Uthus et al. [154, 155], we propose a novel state space formulation on which we apply two classic search algorithms. On the one hand beam search, a heuristic incomplete search algorithm without optimality guarantees, on the other hand weighted A* search, a best-first search algorithm, which is complete and guarantees bounded suboptimality when using an admissible heuristic.

We compare different lower bounds to guide our search, where we solve the arising subproblems either exactly or heuristically. For small to medium-sized instances up to 18 teams, we can precompute a disjoint pattern database to quickly calculate lower bounds for states—admissible heuristics for the search. For larger instances up to 24 teams, we solve the subproblems inexactly resulting in inadmissible heuristics. In a randomized variant of beam search, we employ shuffled team ordering and add small amounts of Gaussian noise to the nodes' guidance for diversification when multiple runs are performed. This allows for a simple yet effective parallelization of the beam search.

A final comparison for beam search is done on the NL, CIRC, NFL, and GALAXY benchmark instances with 12 to 24 teams, for which we report a mean gap difference

to the best known feasible solutions of 1.2% and five new best feasible solutions. In the subsequent chapter, we demonstrate how a large-scale beam search variant with a more sophisticated parallelization and large amounts of memory allows reducing this mean gap difference below zero providing many more new best solutions up to 22 teams (see Appendix A).

The work on beam search for the TTP was first presented at the EvoCOP 2020 conference and published in its proceedings. Afterwards, a subsequent invited extended journal version was published in the Evolutionary Computation Journal:

> Nikolaus Frohner, Bernhard Neumann, and Günther R Raidl. A beam search approach to the traveling tournament problem. In *Evolutionary Computation in Combinatorial Optimization – 20th European Conference, EvoCOP 2020*, volume 12102 of *LNCS*, pages 67–82. Springer, 2020
>
> Nikolaus Frohner, Bernhard Neumann, Giulio Pace, and Günther R Raidl. Approaching the traveling tournament problem with randomized beam search. *Evolutionary Computation Journal*, 2022. in press

The weighted A* search together with a stronger heuristic for the TTP is an additional yet unpublished contribution of this thesis. We show that duplicate state detection and a dynamic team ordering may substantially reduce the A* search effort in terms of expanded nodes and runtime. While we suspect more memory and parallelization are required to improve lower bounds, competitive solutions can be derived for instances up to 14 teams using an anytime variant of this search.

3.1 Introduction

In 2001, Easton, Nemhauser, and Trick introduced the traveling tournament problem (TTP), an $\mathcal{NP}$-hard combinatorial optimization problem in the realm of sports league scheduling. It is concerned with the construction of a double round robin tournament for a sports league, where the sum of the travel distances over all teams shall be minimized. Teams start and end at their respective home venues and are assumed to always travel directly from their current position to their next designated game venue, which is either at home or away. Each team is only allowed to play a certain maximum number of games away or at home consecutively, and two teams must not play against each other in two subsequent rounds. At the time of writing, proven optimal solutions have been found for classical benchmarks instances with up to ten teams, but not for twelve and more teams, as can be checked on the recent RobinX solution repository.[1]

Due to the problem's practical hardness, many different metaheuristics have already been applied to create high-quality but not necessarily optimal solutions in reasonable

[1] `https://www.sportscheduling.ugent.be/RobinX`

time. Neighborhood search based approaches such as tabu search [39] or simulated annealing [4, 160] provide particularly strong results. Inspired by the success of the exact tree search methods of Uthus et al. [154, 155] on small to medium-sized instances, we present in this chapter a variant of the heuristic search algorithm *beam search* based on a state space formulation of the problem. We compare different lower bound based heuristics used to order the nodes in a layer of the state graph, which is traversed in a truncated breadth-first-search manner. These guidance heuristics are calculated by solving corresponding capacitated vehicle routing problems (CVRPs) either exactly, resulting in an admissible heuristic, or heuristically[2], losing the guarantee of admissibility. The presented CVRPs are relaxations of the TTP which are $\mathcal{NP}$-hard themselves.

Competitive results and new best feasible solutions can be achieved on difficult benchmark instances with a randomized variant of the beam search in which we shuffle the team ordering and add Gaussian noise to the heuristic estimates. This randomization enables a simple yet effective parallel execution of multiple diversified beam search runs. We compare the beam search approach with a state-of-the-art simulated annealing approach from the literature [4] and with weighted A* search on the proposed state space formulation. We made our implementation available open source on GitHub.[3]

Structure. We formally introduce the TTP in Section 3.2 and give an associated state space formulation in Section 3.4, where an optimal solution corresponds to a shortest path through a directed acyclic state graph. The same state may be reached by different partial schedules and determines the set of feasible completions, i.e., all the continuations leading to a complete TTP schedule when combined with any of the partial schedules. The approach, therefore, allows detecting isomorphisms to reduce the search effort. In between in Section 3.3, we summarize related work on which our approach is based. Specifically worth mentioning are the papers of Uthus et al. [153, 154, 155], which broadly fall into the class of tree search based techniques. In particular, we build upon their bound precalculation method.

Section 3.5 is concerned with the schedule construction algorithm on the state graph using beam search driven by lower bound based heuristics and infeasibility checks that are derived from the states. These lower bounds are calculated by solving related CVRP instances independently for each team, introduced as *independent lower bound* (ILB) by Easton et al. [46] for the root state and strengthened by Uthus et al. [155] for arbitrary states. We describe in more detail in Section 3.6 how we solve these subproblems either beforehand for all states up to 18 teams exactly via recursive constrained shortest path problems on decision diagrams, or heuristically on the fly with Google OR-Tools[4] up to 24 teams, also providing experimental results.

In Section 3.7, we study the impact of the beam width, reflective symmetry breaking, and different static team orderings on selected instances. In Section 3.8, we describe a weighted

[2]Be aware of the two different meanings of heuristics in this thesis.
[3]https://github.com/nfrohner/ttpbeam
[4]https://developers.google.com/optimization/

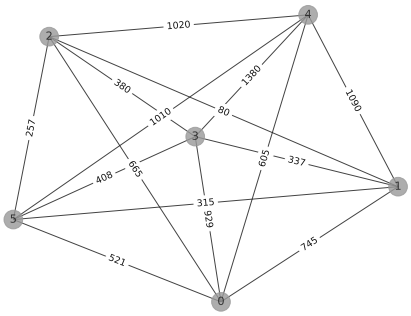
A* approach to the TTP, with the advantage over beam search of being complete and providing bounded suboptimality guarantees. Section 3.9 presents final computational results for beam search, split into experiments with our first implementation in Python [57] and our revised and faster version in Julia [62]. We first compare the performance of different algorithm variants on randomly generated instances on a two-dimensional grid to make final design choices. After that, we present a head-to-head comparison with a reimplemented state-of-the-art simulated annealing approach, where our approach proves to be competitive in a time-limited setting. We conduct final tests on the classical benchmark instances derived from teams of the US Major League Baseball (NL), the national football league instances (NFL), the circular instances (CIRC) by Easton et al. [46], and the instances derived from three-dimensional distances between stars (GALAXY) by Uthus et al. [155], with up to 24 teams. We observe that our constructive approach with a final short best-improvement local search delivers competitive results with a mean gap difference to the best-known feasible solutions of 1.2%. Moreover, we could find new best solutions for four GALAXY instances and one CIRC instance.

We finalize our computational study by a study of weighted A* search in Section 3.10, where we show how to substantially reduce the search effort using duplicate states detection, dynamic team ordering, and a stronger admissible heuristic. Finally, we conclude in Section 3.11 and make suggestions for further research.

3.2 Problem Formalization

We are given a set $V = \{1, \ldots, n\}$ of n teams, where n is even, and a distance matrix d where $d(i, j)$ is the traveling distance from team i's home venue to team j's home venue, $i, j \in V$. A *home stand* (or *away streak*) of a team is a maximal sequence of consecutive games where the team plays always at home (or always away). The goal is to construct a double round robin tournament, where all home stands and away streaks consist of at most U games consecutively at home or on the road (*at-most* constraint), teams do not play against each other in subsequent rounds (*no-repeat* constraint), and the total travel distance over all teams is minimized. Each team starts and ends at its home venue and travels directly between the venues.

Adopting the formulation of De Werra [36] and Ribeiro and Urrutia [124], we see the teams V as vertices of a complete weighted directed graph $G = (V, A)$, where the weights are given by the distance matrix d. The graph is now directed to model the venue of a game, which is represented by a single arc. A double round robin schedule T is an ordered 1-factorization $T = (G^1 = (V, A^1), \ldots, G^{2n-2} = (V, A^{2n-2}))$ of G, which is an ordered partitioning of the arcs into $2n - 2$ perfect matchings (1-factors) $A^r, r \in \{1, \ldots, 2n - 2\}$. An arc (i, j) (or $i \to^r j$) denotes that team i plays against team j at j's venue in round r. The location of team i in round r is denoted $p_i^r \in V$ and determined by the single arc in A^r incident to team i. The objective value of a schedule T is the total travel distance



opponent/venue for team i

in round r

$$\begin{pmatrix} 5 & -3 & 2 & 6 & -1 & -4 \\ -3 & 6 & 1 & 5 & -4 & -2 \\ -2 & 1 & -4 & 3 & -6 & 5 \\ -5 & 3 & -2 & -6 & 1 & 4 \\ 4 & -5 & 6 & -1 & 2 & -3 \\ 2 & -1 & 4 & -3 & 6 & -5 \\ 6 & -4 & 5 & 2 & -3 & -1 \\ -4 & 5 & -6 & 1 & -2 & 3 \\ -6 & 4 & -5 & -2 & 3 & 1 \\ 3 & -6 & -1 & -5 & 4 & 2 \end{pmatrix}$$

Figure 3.1: Left: The NL6 problem instance from [46] shown as complete undirected graph. Right: A feasible double round robin tournament schedule represented by a $(2n-2) \times n$ matrix, where the value j of entry (r, i) corresponds to the game $i \to^r |j|$ if j is negative, otherwise to the game $j \to^r i$.

given by

$$z(T) = \sum_{i=1}^{n} \left(d(i, p_i^1) + \sum_{r=2}^{2n-2} d(p_i^{r-1}, p_i^r) + d(p_i^{2n-2}, i) \right). \tag{3.1}$$

Throughout this paper and as in most previous work, we only consider $U = 3$, for which Thielen and Westphal [144] have shown strong NP-completeness of the corresponding decision variant of the problem. Furthermore, we assume that the distances are symmetric.

Figure 3.1 shows on the left an example instance with $n = 6$ teams depicted as an undirected complete graph. A corresponding feasible TTP schedule is shown on the right, represented as a $(2n-2) = 10$ rounds by six teams matrix, denoting the opponent and venue for each round and team. We denote an abstract opponent of a team's home game by $\mathcal{H}$ and of an away game by $\mathcal{A}$, when only the venue is important.

3.3 Related Work

Van Bulck et al. [157] provide a recent literature overview and classification of round-robin sports timetabling problems, where the TTP belongs to a class encoded as 2RR, C, $\emptyset$ | CA3, SE1 | TR. The first part of the classification defines the structure of the tournament, the second the constraints, and the third the objective function. Another recent review of sports league scheduling and related topics with a focus on Latin America is presented by Durán [44].

The TTP itself, together with the NL and CIRC benchmark instances, and the ILB were introduced by Easton et al. [46]. They also provide two first solution approaches, one mainly based on constraint programming with an enumeration scheme on the increasing number of trips strengthening the independent lower bound, and one on integer programming with a formulation that directly operates on all possible trips. These are limited to solve instances only up to six teams. Easton et al. [47] present a

parallel branch-and-price approach where the columns are the concrete single team tours, combining integer programming with constraint programming for the pricing problem and as a primal heuristic. This approach is able to solve instances with up to eight teams without no-repeat constraints to optimality with 20 processors in a couple of days.

Irnich [88] introduces a compact formulation where the schedules are modeled as combined movements on a time-discrete network for each team. An extensive formulation to be solved with branch-and-price is derived where the separate tours for the teams are modeled on related state graphs where restricted shortest path problems are solved for the pricing problems. Together with reflective symmetry breaking, the NL instance with eight teams (NL8) and no-repeat constraints could be solved for the first time to optimality within 12 hours in single-threaded mode.

Uthus et al. [154] propose a DFS* approach to solve the TTP, guided by heuristic estimates based on the ILB derived from states and using reflective symmetry breaking to solve instances up to eight teams by orders of magnitude faster than previous approaches. More specifically, the authors reported solving NL8 in minutes instead of hours as by Irnich [88]. Pushing it further, Uthus et al. [155] suggest an exact iterative deepening A* search, which allows them to solve benchmark instances with ten teams to proven optimality for the first time and to improve lower bounds for instances with 12 and 14 teams. To achieve this, their approach features special symmetry breaking techniques, memoization, and was executed in parallel using subtree splitting on 120 processors for a wall clock time of a couple of days. From Uthus et al. [154, 155], we adopt and extend the idea to populate disjoint pattern databases [94] by precalculating independent lower bounds for teams' states to occur during the state space traversal.

We aim at solving larger instances heuristically and therefore compete with today's state-of-the-art metaheuristic approaches, which we consider to be the simulated annealing (TTSA) from Anagnostopoulos et al. [4] and its parallel variant, the population-based simulated annealing from Van Hentenryck and Vergados [160] (PBSA). The latter found the so far best solutions for most of the larger NL, NFL, and CIRC instances using a cluster consisting of 60 nodes within a few hours per instance. TTSA starts on a randomly generated solution and makes use of five neighborhood structures. It allows the traversing of infeasible regions of the search space steered by a modified objective function and strategic oscillation. A thorough analysis of these neighborhoods is performed in the tabu search approach by Di Gaspero and Schaerf [39].

On the more constructive side, the ant colony optimization from [153] (AFC-TTP) makes use of constraint propagation, sophisticated home-away pattern generation and matching, and backjumping to favor the creation of feasible solutions. Solution quality guidance is derived from experience encoded into the pheromone matrix without a local heuristic, since it turned out to be misleading. The ants' solutions are improved with a fast tabu search operating in the feasible region of the search space. Goerigk and Westphal [73] propose a hybrid approach with alternating phases of tabu search and integer programming, where the latter in a fix-and-optimize manner either optimizes in the space of opponents per slot with fixed home-away patterns or vice versa, achieving

best feasible solutions for larger GALAXY instances. Goerigk et al. [74] introduce a new schedule construction heuristics based on a minimum-length three-vertex path packing of the team graph determining the teams' away streaks, limited to instances with $n \equiv 4 \mod 6$ teams. This heuristic is combined with the alternating tabu search and integer programming approach from before and achieved new best feasible solutions for large GALAXY and NFL instances.

Beam search can be seen as a constructive metaheuristic, where multiple solutions are created in parallel using a truncated breadth-first search. In each layer of the search, only a certain number of the most promising nodes are pursued further to keep the number of search nodes considered polynomially bounded. It was introduced by Lowerre [105] to find high-quality solutions in a state transition network for a speech recognition system. For beam search with the focus on scheduling see, e.g., Ow and Morton [113]. We model the capacitated vehicle routing problems corresponding to the state-related independent lower bounds as recursive constrained shortest path problems on decision diagrams, a close relative of state graphs, used as a compact data structure to encode solution spaces. For a thorough introduction to decision diagrams in combinatorial optimization, we recommend the book by Bergman et al. [14].

We adopt and extend the formulation of the ILB as a vehicle routing problem from Urrutia et al. [152], in which they present a tightening of the independent lower bound incorporating information of the TTP instances with unit distances (CONST) as solved by Rasmussen and Trick [123]—the *minimum number of trips* (MNT) lower bound. This drops the independency between teams and in some cases enforces them for feasibility reasons to make more tours than would be optimal in an isolated fashion.

Real-world applications of variations of the traveling tournament problem are described by Bonomo et al. [20] for the Argentinian volleyball league and in Durán et al. [45] for the Argentinian basketball league.

3.4 State Space Formulation

We model the solution space, i.e., the set of feasible schedules of a TTP instance (V, d), by a state graph. This is a rooted directed acyclic graph representing the feasible schedules by corresponding paths from a root state to a dedicated terminal state. The states (nodes) are organized into $n^2 - n + 2$ layers, where layer 0 only contains the root state s_r, layer $n^2 - n + 1$ only the terminal state s_t, and layers $l = 1, \ldots, n^2 - n$ contain states representing the situations after the l-th played game.

Each state is a tuple $s = (\mathbf{M}^s, \mathbf{y}^s, \mathbf{r}^s, \mathbf{x}^s, \mathbf{h}^s, \mathbf{o}^s)$, where $\mathbf{M}^s = (M^s_{i,j})_{i,j \in V} \in \{0,1\}^{n \times n}$ is an incidence matrix that indicates the games left to be scheduled and vectors $\mathbf{y}^s = (y^s_i)_{i \in V}$, $\mathbf{r}^s = (r^s_i)_{i \in V}$, $\mathbf{x}^s = (x^s_i)_{i \in V}$, $\mathbf{h}^s = (h^s_i)_{i \in V}$, and $\mathbf{o}^s = (o^s_i)_{i \in V}$ represent for each team i the currently forbidden opponent y^s_i, the current round r^s_i, its location x^s_i, and the number of still possible home or away games left to play in a row h^s_i and o^s_i, respectively. The forbidden opponents $\mathbf{y}^s$ are used to implement the *no-repeat* constraint and $\mathbf{h}^s$ and

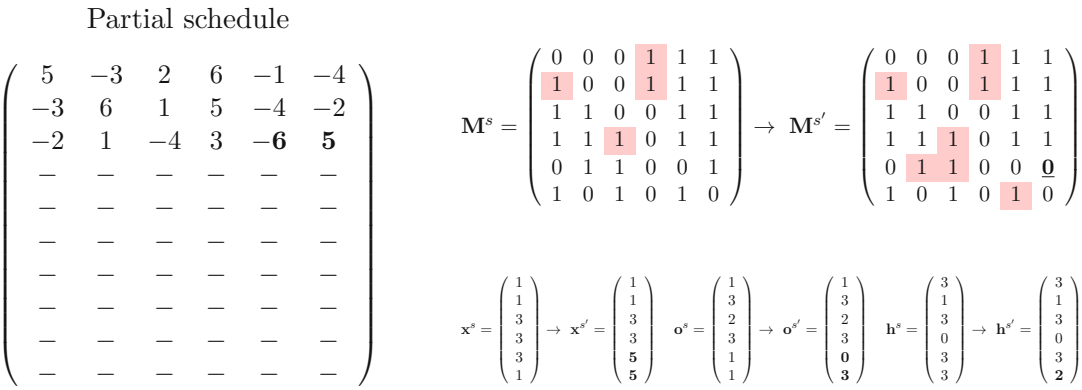


Figure 3.2: Left: Exemplary partial schedule for an instance with six teams before ending the third round, for which the teams six and five (in bold) are selected to play the next game. Right: Corresponding state updates where in the matrix of the games left the currently forbidden games implied by $\mathbf{y}^s, \mathbf{o}^s, \mathbf{h}^s$ are grayed out, changed elements are set in boldface (and underlined for the games matrix). We omitted $\mathbf{r}^s$ and $\mathbf{y}^s$ for space reasons.

$\mathbf{o}^s$ to take care of the *at-most* constraints. Moreover, the information contained in a state implies for each team $i \in V$ the set P_i^s of the games that can be played next without violating the TTP constraints.

A state transition from a state s at layer l to a state s' at layer $l+1$, $l = 0, \ldots, n^2 - n$, corresponds to a specific game $i \rightarrow^r j$ being played by teams i and j at j's venue in round r. Each state transition is weighted by the sum of the distances both teams have to travel from their previous locations to play the game

$$\Delta z(s, s') = d(x_i^s, x_i^{s'}) + d(x_j^s, x_j^{s'}). \tag{3.2}$$

Teams for the game are selected in a way that the partial schedule grows round by round in ascending order where each round is completed before the next one starts. All paths starting from the root state and leading to the terminal state correspond to feasible solutions. Paths that end before the terminal state at a state without further transitions represent partial schedules that cannot be feasibly continued. A shortest path from the root to the terminal state, therefore, corresponds to an optimal feasible solution for a given problem instance.

We introduce two special rounds $r = 0$ and $r = 2n - 1$ where every team is at its home location. Let $\mathbf{M}^{s_\mathrm{r}}$ be the matrix with non-diagonal ones and diagonal zeros, corresponding to all games to be played, and matrix $\mathbf{M}^{s_\mathrm{t}}$ be the all-zeros matrix. If there is no forbidden opponent for a team $i \in V$ in state s, then y_i^s is set to -1. The root state is then $s_\mathrm{r} = (M^{s_\mathrm{r}}, \mathbf{y}^{s_\mathrm{r}} = (-1, \ldots, -1), \mathbf{r}^{s_\mathrm{r}} = (0, \ldots, 0), \mathbf{x}^{s_\mathrm{r}} = (1, \ldots, n), \mathbf{h}^{s_\mathrm{r}} = (U, \ldots, U), \mathbf{o}^{s_\mathrm{r}} = (U, \ldots, U))$ and the terminal state $s_\mathrm{t} = (\mathbf{M}^{s_\mathrm{t}}, \mathbf{y}^{s_\mathrm{t}} = (-1, \ldots, -1), \mathbf{r}^{s_\mathrm{t}} = (2n-1, \ldots, 2n-1), \mathbf{x}^{s_\mathrm{t}} = (1, \ldots, n), \mathbf{h}^{s_\mathrm{t}} = (0, \ldots, 0), \mathbf{o}^{s_\mathrm{t}} = (0, \ldots, 0))$. Transitions to the terminal state are special in the sense that they do not correspond to a played game but represent the return of the teams which are on the road in the last round to their respective home venues.

Transitions from a state s at some layer l to a subsequent state s' at layer $l+1$ are done by selecting a game $(i,j) \in P_i^s$ (or (j,i)) where we impose the condition $r_i = r_j = \min_{k \in V} r_k$. This ensures that the teams are in the same round and games are assigned to teams round by round. We denote the layer of a state by layer(s). If there exists a *dead* team i with $P_i^s = \emptyset$, our current state has no feasible completion. Since there is no meaning in which order we select the teams within a specific round r, we break this symmetry by defining a specific team permutation $\pi \colon V \to V$. At each state of layer l, a game from $P_{\pi_i}^s$ has to be played for which i and r_{π_i} are minimal. A trivial ordering π is the lexicographic ordering of the teams.

Selecting a game (i,j) yields state s' with $\mathbf{M}^{s'}$ being a copy of $\mathbf{M}^s$ except that $M_{i,j}^{s'} = 0$, which implicitly removes this game from $P_i^{s'}$ and $P_j^{s'}$ as well. The position, round, and streak-related information of i and j are updated from s to s' accordingly. To respect the *no-repeat* constraints, the forbidden opponent vector $\mathbf{y}^s$ is copied to $\mathbf{y}^{s'}$ except that $y_i^{s'} = j$ and $y_j^{s'} = i$, if $M_{j,i}^s = 1$; otherwise these values are set to -1. For every other team $k \in V \setminus \{i,j\}$, $y_k^{s'} = -1$ is set if $y_k^s \in \{i,j\}$. The *at-most* constraints are already implied by the updates in $\mathbf{o}^{s'}$ and $\mathbf{h}^{s'}$. If $o_i^{s'} = 0$, then away games are not allowed in the next round for team i; analogously, a continuation of j's home stand is not allowed, if $h_j^{s'} = 0$.

An exemplary state transition is shown in Fig. 3.2 for an instance with six teams before and after ending the third round with the game $(5,6)$. We see that team five hits its away streak limit and all its away games are not available for the next round and that the game $(6,5)$ is forbidden.

3.5 Beam Search

We perform a layer-by-layer breadth-first-search traversal of the state graph, where for each state all permitted games for a selected team are played by performing the respective transitions to corresponding successor states. The current shortest path value and the corresponding partial schedule are cached for each state during construction and updated if a shorter path to an already visited state is discovered.

Due to the complexity of the problem, only instances with four teams admit in practice a complete construction of the state graph, providing a guaranteed optimal solution. We, therefore, restrict the search to an incomplete beam search (see Section 2.2) where at each layer at most β states are kept for further consideration; parameter β is hereby called the beam width. In this way, the total number of expanded states is polynomially bounded by $\mathcal{O}(n^2\beta)$. A shortest path through such a restricted state graph then corresponds to a feasible heuristic solution. To guide the search, in each layer the β most promising states are kept according to some state ranking heuristic $b(s)$, in the hope that the finally shortest path corresponds to an optimal or close-to-optimal solution. Classical beam search sorts the states by an f-value known from A* search that combines the length of the currently shortest path $g(s)$ to the state s with a heuristic estimate $b(s)$ for a best

further continuation to the terminal state:

$$f(s) = g(s) + b(s) \tag{3.3}$$

As tie breaker, we use the shortest path length to the state $g(s)$.

In our beam search implementation, we only keep the current layer in a queue and the successive layer in a priority queue sorted according to f that contains at most the β best successor states so far. The current layer's queue is sorted once by the f-value and iterated from smallest to largest. The priority queue for the successor states is implemented by a max-heap with a bounded number of items combined with a hash map to access arbitrary states in expected constant time. Before creating a successor state, we check in case of a full beam utilizing incremental evaluation whether the potential new state's f-value is worse than the worst f-value in the heap. If this is the case, we do not need to consider the state further. Otherwise, we create the successor state and check whether it already exists in the max-heap, in which case we conditionally update its shortest path value and current best partial schedule. If the state was not yet contained in the heap, we replace its so far worst state with the newly created successor state. This approach gives us a smaller memory footprint than storing all created states until termination, allowing us to work with higher beam widths. The current partial schedule is cached along each state in a growing vector.

As will be discussed in detail in Section 3.6, the heuristic estimates are also cached along the state for each team, together with the number of home and away games left for each team. The latter allows quickly checking whether or not there are enough home games in relation to away games to make a feasible completion.

To diversify the search, we further introduce a randomized variant of the beam search. We apply it in a multi-start fashion for a simple yet effective parallelization of independent workers. To this end we add a normally distributed random offset with standard deviation σ to each state's original f-value:

$$\tilde{f}(s) = f(s) + \mathcal{N}(0, \sigma). \tag{3.4}$$

The motivation is that states which would be pruned when just considering their deterministic f-value get a chance to survive, and they may lead to superior solutions. Initially promising states can also get cut off early by drawing a too-high random offset. It also acts as a random tie breaker for states with the same f-value. This approach has similarities to stochastic ranking, where a stochastic bubble search is applied to rank a population, see [131].

Crucial is the standard deviation σ for which we make the following parameterized ansatz:

$$\sigma = \sigma_{\text{rel}} \cdot b(s_{\text{r}}) \tag{3.5}$$

Parameter σ_{rel} thus determines the fraction of the heuristic estimate of the root state to be used as σ, so that the order of magnitude of the expected solution length of a given

Algorithm 3.1: Randomized beam search for the TTP.

Input: number of teams n, distance matrix d, start state s_{start}, terminal state s_{t}, noise parameter σ_{rel}, state heuristic estimate function b, beam width β
Output: feasible schedule T

```
1  queue Q ← {s_start};
2  for l ← layer(s_start) + 1 to n^2 − n do
3      H ← empty maximum heap;
4      while Q ≠ ∅ do
5          s ← Q.pop;
6          t ← next-team(l, s);
7          foreach (i, j) ∈ {(i', j') ∈ P_t^s | r_{i'}^s = r_{j'}^s} do
8              ε ← N(0, σ_rel · b(s_start));
9              if feasibility-and-optimality-check(H, β, s, b, ε, (i, j)) then
10                 s' ← copy s and make transition by playing (i, j) and updating
                       state along with cached data accordingly;
11                 s'.current_schedule ← s'.current_schedule ∪ (i, j);
12                 f(s') ← g(s') + b(s') + ε;
13                 include s' into H respecting f(s');
14                 if H.size > β then
15                     remove worst element of H;
16                 end
17             end
18         end
19     end
20     Q ←sorted-by-f-value(H);
21 end
22 if Q ≠ ∅ then
23     create going home transitions for all states Q to s_t;
24     return s_t.current_schedule;
25 else
26     return no feasible schedule found;
```

instance is respected. Tuning of this parameter is performed later in the computational study.

Algorithm 3.1 shows our beam search in pseudo-code. In our setting, the starting state s_{start} is the root state s_{r} as defined in Section 3.4, but more generally it could be a state corresponding to an already non-empty partial solution. Function next-team(l, s) selects the team to consider for a given layer l and state s. Trivial options are to take the lexicographically smallest team that is in a minimal round or to initially fix a random permutation of the teams. Further static orderings are considered and evaluated on

artificial tuning instances in Section 3.7.

The computational complexity of a state transition implemented in a straight-forward way is in $\mathcal{O}(n^2)$ as the games matrix and the solution vector are copied and modified. Since n is rather small (≤ 40) and contiguous memory copying is a fast operation, this is not a bottleneck in our implementation.

Procedure feasibility-and-optimality-check$(H, \beta, s, b, \varepsilon, (i, j))$ incrementally checks whether the transition would lead to a state for which we know for sure that it does not have a feasible completion or for which the f-value would be worse than the currently worst in a full beam. Regarding feasibility, we consider the cases when not enough home or away games are available for a specific team to not violate the *at-most* constraint, the only two remaining games for a team would be against one another, violating *no-repeat* for certain, or because one team has an empty possible games set in this round, called *dead teams check*. The first two checks are performed in constant time, while the last one is in our implementation the most expensive check requiring time $\mathcal{O}(n^2)$ since we iterate over all teams and their remaining games to find a team without a permitted game. We speed this up in the implementation by caching the number of teams that just hit the streak limit to make a worst-case estimate per team in constant time regarding the number of forbidden games. If this value is less than the number of the team's remaining games, we can immediately proceed to a next team, without iterating over its specific games. The impact of the dead teams check on the runtime and solution quality is studied in Section 3.9.

The optimality check is done by considering the increase in the f-value by the move and if it is worse than the maximum f-value in a full, i.e., containing β states, max-heap H. In this case, the transition for game (i, j) does not need to be considered and state s' is not created, which saves a costly state expansion operation that would require copying the whole current state and cache variables. After all successor states have been checked and potentially added to the heap H, they are transferred to queue Q, sorted according to state priorities, and thus these nodes become the new current layer. The sorting is done to fill the beam earlier with likely better states to increase the odds of rejecting the creation of successor states during incremental evaluation, and for the duplicate states filtering as described before.

In the next section, we study in detail the crucial part of devising and efficiently calculating a heuristic estimate $b(s)$ for a state s based on the independent lower bound to ultimately determine the state's f-value.

3.6 Lower Bound Based Heuristics

To guide the beam search, we use the f-value of a state s as defined in Equation (3.3), consisting of the shortest path length to the state so far $g(s)$ plus a heuristic estimate $b(s)$. The latter is derived from a lower bound over the feasible completions from state s, either determined exactly or heuristically. We first consider three bounds of different strengths

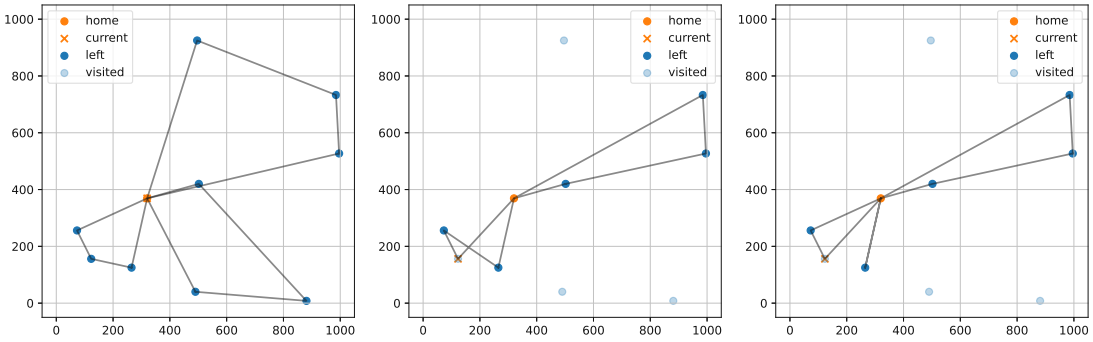


Figure 3.3: Imagine an instance with ten teams and two similar states where a team has already played four away games, being currently away, and either two or three home games. In the latter case, six home games are left, for which at least two home stands ($U = 3$) are necessary, corresponding to the constraints of having at least two tours, an optimal solution is shown in the middle (objective value: 2252). In the former case, seven home games are left and therefore one more home stand is required, an extra tour, which results in a worse optimal solution shown on the right (objective value: 2542). On the left, an optimal solution to the team's root state is shown.

all based on the main idea to consider all teams and their away games independently, as proposed by Easton et al. [46]. These bounds are the Traveling Salesperson Problem (TSP) bound $b^{\text{TSP}}(s)$, the Capacitated Vehicle Routing Problem (CVRP) bound $b^{\text{CVRP}}(s)$, and the CVRP-Home (CVRPH) bound $b^{\text{CVRPH}}(s)$.

Each bound is calculated by accumulating the solution lengths of independent subproblems for the teams. The TSP bound drops the *at-most* constraint, which is considered by the CVRP bound. The latter, however, ignores the minimum number of home stands required (induced by the number of away games remaining) and the maximum number of home stands allowed (induced by the number of home games remaining), which are respected by the last and tightest one, the CVRPH bound. Therefore

$$b^{\text{TSP}}(s) \leq b^{\text{CVRP}}(s) \leq b^{\text{CVRPH}}(s). \tag{3.6}$$

Exact bounds calculation. The main idea for obtaining lower bounds is to relax the problem by considering the tours of all teams independently. Easton et al. [46] already suggested the independent lower bound (ILB) that applies this principle. This bound neglects the home games and the *no-repeat* constraints and considers only the away games for a given team $i \in V$ with only the away *at-most* constraints. This amounts to a capacitated vehicle routing problem (CVRP), also formulated as such by Urrutia et al. [152], where the depot is at i's home venue, the customers are the away teams with unit demand, and the vehicles' capacities are $U = 3$. The CVRP itself is strongly $\mathcal{NP}$-hard but for few customers tractable in practice.

Given an arbitrary state s and team i, we have to consider the remaining away teams $\mathcal{A}_i^s$ for i, the position x_i^s, and the remaining away streak o_i^s. If $x_i^s \neq i \wedge o_i^s = 0$, then we consider an artificial state in which the team is assumed to have returned home (this is the only option it has at that moment), $o_i^s = \min(\mathcal{A}_i^s, U)$, and add $d(x_i^s, i)$ to the resulting bound. Let the optimal total length for this problem for team i be $b_i^{\text{CVRP}}(s)$.

Then, the sum of the optimal values over all teams is a lower bound for the optimal value of the corresponding TTP-feasible completion of s

$$b^{\text{CVRP}}(s) = \sum_{i=1}^{n} b_i^{\text{CVRP}}(s). \tag{3.7}$$

A natural further relaxation is to drop even the away *at-most* constraints, and a traveling salesperson problem based lower bound b^{TSP} emerges, if the triangle inequality holds, otherwise it relates to a VRP with unlimited capacity. In both cases, we do not have to consider the current away streak of team i in state s.

To provide better guidance and pruning power for our search algorithms, we are more interested in tighter lower bounds while keeping their computational costs in mind. A first natural strengthening is to consider also the home *at-most* constraints. Let $h_i^{\text{left}} = |\mathcal{H}_i^s|$ be the number of home games left for team i in state s. Then we need at least $\tilde{h}_i^{\min} = \lceil (h_{\text{left}} + \bar{h}_i)/U \rceil$ home stands to accommodate for the home games, where $\bar{h}_i$ is the length of the current home stand. Translated to the CVRP, this amounts to the constraint that we need to perform at least $\tilde{h}_i^{\min}$ non-trivial tours (after every away streak comes a home stand), minus one if the team is currently at home. Analogously, every away streak (tour) needs at least one remaining home game from where it came (except for the first tour), i.e., we can have at most $1 + h_i^{\text{left}}$ tours. This gives us a maximum to the home stands $\tilde{h}_i^{\max} = 1_{x_i^s = i} + 1 + h_i^{\text{left}}$ we can realize from a given state.[5] We can therefore define a home stand constrained lower bound $b_i^{\text{CVRPC}}(s, \tilde{h})$ and tighten the CVRP bound by finding the minimum within the range of allowed home stands, summed over all teams resulting in the CVRP with home stands bound (CVRPH)

$$b^{\text{CVRPH}}(s) = \sum_{i=1}^{n} \min_{\tilde{h} \in \{\tilde{h}_i^{\min}, \ldots, \tilde{h}_i^{\max}\}} b_i^{\text{CVRPC}}(s, \tilde{h}). \tag{3.8}$$

In Figure 3.3 an effect of these constraints on an example instance with ten teams is illustrated. If a $b_i^{\text{CVRPC}}(s, \tilde{h})$ has no feasible solution, then we set it to ∞. To exclude them in the minimization, we can strengthen the lower and upper bound to $\tilde{h}$ further, by inferring the minimum number of required and maximum number of possible tours (and therefore home stands) based on the remaining away games, with analogous arguments as before for the remaining home games.

Disjoint pattern database. In principle we could evaluate any given state by feeding a corresponding integer programming (IP) model into a solver like Gurobi[6], but this is expected to be too costly in practice. To speed up our search algorithms, we pre-calculate the lower bounds for the states that can occur for a given TTP instance, similarly as done

[5] We denote by 1_{expr} the indicator function, which maps to 1 when the subscripted expression holds, otherwise to 0.

[6] https://www.gurobi.com/

by Uthus et al. [155]. We do this by representing the whole space of feasible solutions to the given CVRP instance for each team i with an exact multi-valued decision diagram (DD) [14] and finally store the lower bounds for the states that occurred in a lookup table, which can then be used as disjoint pattern database [94, 155] during the search. Each node in this DD is associated with a state q consisting of the away games left to play $\mathcal{A}^q$ (represented by the subset of other teams against which team i still has to play), the team i's position x^q and the current number of consecutive away games, the away streak $\bar{o}^q$. The root state for a given team i is therefore $q_\mathrm{r} = (\{1, \ldots, i-1, i+1, \ldots, n\}, i, 0)$. Transitions are made until the terminal state $q_\mathrm{t} = (\{\}, i, 0)$ is reached, where in every layer, all available transitions are performed. Hereby we distinguish between three possibilities:

- Select any away team left $j \in \mathcal{A}^q$ to visit next if there are such, and go home afterwards, where costs $d(x^q, j) + d(j, i)$ accrue. The state is updated accordingly to $(\mathcal{A}^q \setminus \{j\}, i, 0)$.
- If $\bar{o}^q$ is less than $U - 1$, then select any away team left $j \in \mathcal{A}^q$ to visit next, if there are such, and stay at j afterwards, where costs $d(x^q, j)$ accrue. The state is updated to $(\mathcal{A}^q \setminus \{j\}, j, \bar{o}^q + 1)$.
- If $\mathcal{A}^q$ is empty, go home if not already at home, where costs $d(x^q, i)$ accrue. The state is updated to the terminal state q_t.

Paths from the root to the terminal node in the DD then correspond to the feasible solutions of the CVRP, and with the costs associated with the transitions (i.e., arcs in the DD), the lengths of such paths correspond to solution lengths. For each node in the CVRP decision diagram, the shortest path to the terminal node is calculated and saved in the lookup table, serving as a lower bound for a team with given away teams to play, being at a position either at home or at some away team and its current away streak. Being a layered directed acyclic multigraph, the shortest paths for each node in the decision diagram can be calculated efficiently by doing a breadth first search backwards from the terminal to the root node.

The TSP-based bound values can also be pre-calculated by this method by simply ignoring the away streak and allowing always a direct transition to a next away team without going home first.

Furthermore, for the CVRPH bound, we consider constrained shortest path lengths $z^{\mathrm{sp}}(q, \tilde{h})$ from any node to the terminal node, with the constraint that exactly $\tilde{h}$ home stands occur. This means that at most $\tilde{h}U - \bar{h}_i$ home games can be played from a given node, where $\bar{h}_i$ is the length of the current home stand for team i. At the terminal node $z^{\mathrm{sp}}(q_\mathrm{t}, 1) = 0$ and ∞ for every other node. In the backward sweep from q' to q with arc costs $c_{q,q'}$, if $x_q \neq i$, then we set $z^{\mathrm{sp}}(q, \tilde{h}) = \min\{z^{\mathrm{sp}}(q', \tilde{h}) + c_{q,q'}, z^{\mathrm{sp}}(q, \tilde{h})\}$. If on the other hand $x^q = i$, i.e., a new home stand has occurred, we set $z^{\mathrm{sp}}(q, \tilde{h}+1) = \min\{z^{\mathrm{sp}}(q', \tilde{h}) + c_{q,q'}, z^{\mathrm{sp}}(q, \tilde{h}+1)\}$. For each state, we now have all the constrained lower bound values available that correspond to $b_i^{\mathrm{CVRPC}}(s, \tilde{h})$. Additionally, we define

$b_i^{\text{CVRPC},\geq}(s,\tilde{h}) = \min_{\tilde{h}' \in \{\tilde{h},\ldots,\tilde{h}_i^{\max}\}} b_i^{\text{CVRPC}}(s,\tilde{h}')\ \forall \tilde{h} \in \{\tilde{h}_i^{\min},\ldots,\tilde{h}_i^{\max}\}$, which gives us the lower bounds when using *at least* $\tilde{h}$ home stands.

In the lookup for the CVRPH bound of a single team, the minimum over a sequence is fetched, defined by the minimum and maximum number of home stands determined by feasibility and optimality considerations of the team's state. For instance, the number of remaining home games may require a certain number of home stands to play them, while performing more than one length-one tour is never strictly optimal for a capacitated vehicle routing problem with a capacity greater than one, providing a tighter maximum number of home stands. The latter is true if the triangle inequality holds and therefore merging two length-one tours never increases the traveling distance. Furthermore, since we do not model abstract home games explicitly in our state graph, we need to take the minimum of going home first or directly continuing a current away streak. The slight increase in lookup complexity allows for a reduction of the state space complexity.

Dropping independence.† So far, we had to solve independent subproblems and made use of the disjoint pattern database to quickly calculate an admissible heuristic for TTP states. Single team strengthening constraints allowed to further tighten the bound which can be modeled as a *joint*—all teams have their own vehicles and remaining away games—CVRP with separate lower and upper limits for the number of vehicles.

By dropping this independence we can add further strengthening constraints based on rules derived from feasibility considerations on how streaks can be continued. We assume a TTP state s and consider a team i which is currently away and has a set of away games left $\mathcal{A}_i^s$. If it were to continue its streak by visiting opponent $\omega_i \in \mathcal{A}_i^s$ away, we know that (i,ω_i) is i's next game in the schedule and (ω_i,i) is ω_i's game in the corresponding round. The set of possible opponents to continue i's streak can be reduced to $\mathcal{A}_i'^s$ by removing opponents ω, against which it is currently not permitted to play, for which we know the following reasons:

- Opponent ω is one round ahead of i, i.e., has already played in this round.
- Opponent ω is in the same round, currently at home, and has reached the home streak limit U.

The dependence on the global TTP state remains static, the joint CVRP still consists of n independent ones, one CVRP for each team. This changes with further linking constraint, for all teams i:

- If $\omega_i = j$ and $x_j^s \neq j$, then $\omega_j = \emptyset$, i.e., if opponent j is away, it cannot continue its streak, since it has to return home to play against i.
- If $\omega_i = j$, then $\omega_k \neq j\ \forall k \neq i$: $x_k^s \neq k \wedge r_k^s = r_i^s$, i.e., no two teams in the same round can continue their current away streak with the same opponent.

We call the resulting problems and related bounds derived from optimal solutions Joint-CVRPH (JCVRPH, containing only the static constraints), and JCVRPH2 (containing also the linking constraints) to emphasize the connected CVRPs have to be solved jointly. We formulate them as three-index IP models with decision variables x^{arc}_{ijk}, costs c_{ijk} based on the venue distances, and unit demands for each team's remaining away games, and depots at the team venues. A decision $x^{\text{arc}}_{ijk} = 1$ indicates that team i moves directly from venue j to venue k in some tour. If a team i is currently away at x_i, its current position is also added to the model with a demand equal to the streak length, forcing $x^{\text{arc}}_{iix_i} = 1$, and setting $c_{iix_i} = 0$. The outgoing arcs $x^{\text{arc}}_{ix_ik}$, $x_i \neq i$ need special treatment in terms of additional constraints since they decide how team i's current streak will be continued.

The JCVRPH can also be solved rather quickly by using the single team instances' decision diagrams/pattern databases and summing over each team's minimum over the allowed outgoing arcs of the current state (costs plus lower bound to goal state).We believe JCVRPH2 can be reduced to a minimum-weight matching problem and solved directly without an IP, but leave this as future work.

Minimum number of trips bound. For an even further tightening of the CVRPH and the JCVRPH bounds, we make use of the minimum number of trips (MNT) bound by Urrutia et al. [152]. As the JCVRPH bounds, it does not assume strong independence between the teams anymore. Instead, the relaxed CONSTANT variant of the problem, where all distances are set to one is solved to optimality (or taking a lower bound), yielding a minimum number of trips all teams together have to perform in a feasible solution to the problem. A trip in this case is an atomic movement of a team from one venue to another. Given a state s, let us call $\tau = \sum_{i=1}^{n} t_i$ the number of trips performed so far by all teams in the shortest path from s_{r} to s, where τ is now also part of the state. By the CVRPC bound, each team has an optimal number $\tilde{h}_i^{\text{opt}}$ of home stands from s to s_{t}. This translates to an optimal number of trips $t_i^{\text{opt}} = |\mathcal{A}_i^s| + \tilde{h}^{\text{opt}} - 1_{x_i=i}$. Let τ^{lb} be the lower bound for the minimum number of trips. If $\tau^{\text{lb}} \leq \tau + \sum_i t_i^{\text{opt}}$, then we cannot tighten the CVRPH bound further. Otherwise, we can add the constraint that the teams have to perform $\Delta\tau = \tau^{\text{lb}} - \tau - \sum_i t_i^{\text{opt}}$ extra trips, yielding the MNT bound

$$b^{\text{MNT}}(s) = \min \quad \sum_{i=1}^{n} b_i^{\text{CVRPC},\geq}(s, \tilde{h}_i) \tag{3.9}$$

$$\text{s.t.} \quad \tilde{h}_i \in \{\tilde{h}_i^{\min}, \ldots, \tilde{h}_i^{\max}\} \quad \forall i \in 1, \ldots, n \tag{3.10}$$

$$\tau + \underbrace{\sum_{i=1}^{n} |\mathcal{A}_i^s|}_{|P^s|} + \tilde{h}_i - 1_{x_i=i} \geq \tau^{\text{lb}} \tag{3.11}$$

This bound can be calculated by solving a corresponding integer linear program using binary decision variables $y_i^{\tilde{h}}$ with costs derived from the CVRPC lower bound function

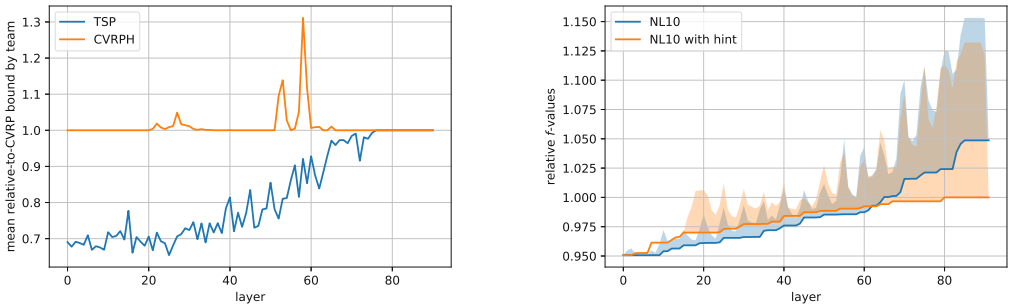


Figure 3.4: Left: Mean bounds by team relative to the CVRP bounds over layers for two exemplary single beam search runs with beam width 1 000 on the classic NL10 instance, either guided by the TSP bound or by the CVRPH bound. The TSP bound is as expected quite weak, the CVRPH bound is most of the time identical to the CVRP bound but provides sometimes substantially more strength. Right: In blue, the evolution of the minimum and the spread of the relative f-values over nodes by layer for a beam search run on NL10 with beam width 1 000 guided by the CVRP bound. We observe that after two-thirds of the layers, the f-values' minimum and spread increase substantially, where the possibilities are already quite constrained by the earlier decisions and bad decisions become necessary more likely. In orange, we additionally hint the first three rounds (15 games) by a known optimal solution, leading to this optimal solution when completing this stub by beam search.

$c_i^{\tilde{h}}$ and counting values $d_i^{\tilde{h}} \in \{\tilde{h}_i^{\min}, \ldots, \tilde{h}_i^{\max}\}$:

$$b^{\mathrm{MNT}}(s) = \min \quad \sum_{i=1}^{n} c_i^{\tilde{h}} y_i^{\tilde{h}} \tag{3.12}$$

$$\text{s.t.} \quad \sum_{\tilde{h}} y_i^{\tilde{h}} = 1 \quad \forall i \in 1, \ldots, n \tag{3.13}$$

$$\sum_{i,\tilde{h}} y_i^{\tilde{h}} d_i^{\tilde{h}} \geq \tau^{\mathrm{lb}} - |P^s| - \tau + \sum_{i=1}^{n} 1_{x_i=i} \tag{3.14}$$

We take the τ^{lb} values from [123], where Rasmussen and Trick present a Benders decomposition approach to solve the CONSTANT instances for up to 16 teams each within at most five minutes. A constraint on the lower trips can also be directly incorporated into a JCVRPH(2) IP model by a lower bound on the active arcs.

Illustrative bounds comparison. The performance of the fastest bounds[7] is exemplarily illustrated in Figure 3.4. On the left, we compare two different beam search runs with beam width 1 000 on TTP benchmark instance NL10 guided by either the TSP or the CVRPH bound. In each layer mean bounds *per team* relative to CVRP bounds are presented. The TSP bound is as expected rather weak and the CVRPH bound is often the same as the CVRP bound but provides additional strength at certain parts of the search. On the right with the same configuration and guidance by the CVRP bound, the minimum and spread of the f-values over the layers are plotted. As commonly observed for greedy approaches (e.g., the nearest neighbor heuristic for the traveling salesperson

[7] the JCVRPH bound was developed later and is therefore not part of the beam search experiments

Table 3.1: Comparison of independent lower bound calculations on randomly generated Euclidean instances with $\{18, 20, 22\}$ teams, 30 instances each, Gurobi 8.1 vs. the *local cheapest arc* construction heuristic and subsequent *greedy descent* from Google OR-Tools 7.5. $\widetilde{t_{\text{opt}}}$ is Gurobi's median time in seconds to prove optimality, $\overline{t_{\text{OR}}}[s]$ the mean time in seconds to obtain the heuristic solution, and $\overline{u_{\text{OR}}^{\text{rel}}}$ are the corresponding optimality gaps with standard deviations.

class	$\widetilde{t_{\text{opt}}}[s]$	$\overline{t_{\text{OR}}}[s]$	$\overline{u_{\text{OR}}^{\text{rel}}}$
$\mathcal{I}_{L^2}^{18}$	865	0.9	0.011 ± 0.007
$\mathcal{I}_{L^2}^{20}$	1256	1.2	0.011 ± 0.007
$\mathcal{I}_{L^2}^{22}$	37183	1.6	0.012 ± 0.005

or the Hamiltonian cycle problem), seemingly good decisions in the beginning may lead to a decline in performance deeper down in the construction, where costly choices are forced to ensure feasibility. Steep relative increases of the f-values after two-thirds of the beam search layers are a manifestation of this effect. We deem the provided example for the TTP instructive where only a rather small part of the solution needs to be fixed—the first three or 1/6 of the rounds—to a known optimal solution as a seed for the beam search, which is then able to continue it to an optimum. In this case, sacrifices are made in the beginning which eventually pay off. While the general pattern of the spread is similar, we also observe that the maximum f-value within a layer increases more rapidly above a value of one, corresponding to states that do not lead to an optimum for sure.

Heuristic calculation. CVRPH bounds for instances with 18 teams could be pre-calculated by our Julia 1.4 implementation on an Intel Xeon E5-2640 processor with 2.40 GHz in single-threaded mode and 32 GB memory in about 70 minutes each, for smaller instances within 10 minutes each. Starting from 20 teams, calculating the CVRPH bounds for all team states in advance becomes intractable due to the growth of the number of bounds which is in $\mathcal{O}(n^3 2^n)$. A key observation is that the number of node expansions in the beam search is polynomially bounded, and so is the number of lower bounds used. The idea now is to calculate the desired bounds not beforehand but on the fly and keep them in a global cache. One transition in the beam search concerns only two teams, where the bounds for the rest can be used from the cache. Also, when the same single team state is reached from a different source TTP state, the value can be retrieved from the global cache. Preliminary results show a cache efficiency $\geq 99\%$. Still, the absolute number of calculations is substantial and the on-the-fly calculation needs to be fast which leads us to a heuristic $\tilde{f}$-function

$$\tilde{f}(s) = g(s) + \hat{b}(s), \tag{3.15}$$

where $\hat{b}(s)$ is a heuristically obtained upper bound to the corresponding lower bound $b(s)$. We refer to the heuristic versions of our bounds by prepending "Heuristic" to the exact bound's name, e.g., Heuristic CVRP (HCVRP) and Heuristic CVRPH (HCVRPH).

As first empirical motivation, we compare in Table 3.1 the exact independent lower bound

calculated using Gurobi 8.1[8] for the root states of 90 randomly generated instances with respective upper bounds obtained by a *local cheapest arc* construction heuristic and a subsequent *greedy descent* from the Google OR-Tools 7.5. We observe promising mean optimality gaps of about 1% with little variability achieved within seconds, while for the 22 teams instances proving optimality takes hours.

The CVRP bound for a single team in a given state is modeled in Google OR-Tools by providing it with the distance matrix of the away games left for a team and the team's home venue as depot. Capacity constraints of the TTP streak limit of U are employed for the vehicles and a demand of one is set for each away team. If the team is currently away, its position is also added to the problem, where the distance from the depot is set to zero. The current streak is set as a demand, in our case either one (two more teams possible in team's away streak) or two (one more team possible in team's away streak). To solve the CVRPH bound problems heuristically, which sometimes strengthens the CVRP bound since it takes required home stands into account, additional constraints are added to use a minimum and maximum number of home stands. We have therefore two different models, one for HCVRP (arbitrary number of vehicles) and one for HCVRPH (bounds on number of vehicles), for which we observe the Google OR-Tools solver to behave differently.

To compare different combinations of construction heuristic (aka first solution strategy) and improvement heuristic (aka search strategy) in Google OR-Tools, we created the aforementioned artificial TTP instances on a 1000×1000 integer grid with uniform spatial distribution and rounded Euclidean distances with 18, 20, and 22 teams, 30 instances each, resulting in 90 instances in total. For each, we solved the independent lower bound to optimality using Gurobi 8.1. In the following experiment, we compare the performance and running times of different combinations of construction and improvement heuristics. The performance is measured as the relative gap between the heuristic solution and the independent lower bound that we previously calculated.

The construction heuristics tested are *path cheapest arc*, *path most constrained arc*, *local cheapest insertion*, *local cheapest insertion*, *parallel cheapest insertion*, *global cheapest insertion*, *first unbound min value*, and *Christofides*. The improvement methods under consideration are *greedy descent*, *guided local search*, *simulated annealing*, and *tabu search*. *Local cheapest arc* is excluded since it becomes very slow for larger instances with the HCVRPH model. As a trade-off between runtime and solution quality deduced from preliminary tests, we set the solution limit to 30 except for *greedy descent* which we let converge to a local optimum. Otherwise, we use for all search methods the default configuration of Google OR-Tools, which we assume to be already sensible for solving vehicle routing problems. The results of the potentially tighter HCVRPH model for instances with 18 and 22 teams are shown in Figure 3.5, presented as boxplots for relative gaps and runtimes over all configurations.

[8] https://www.gurobi.com/

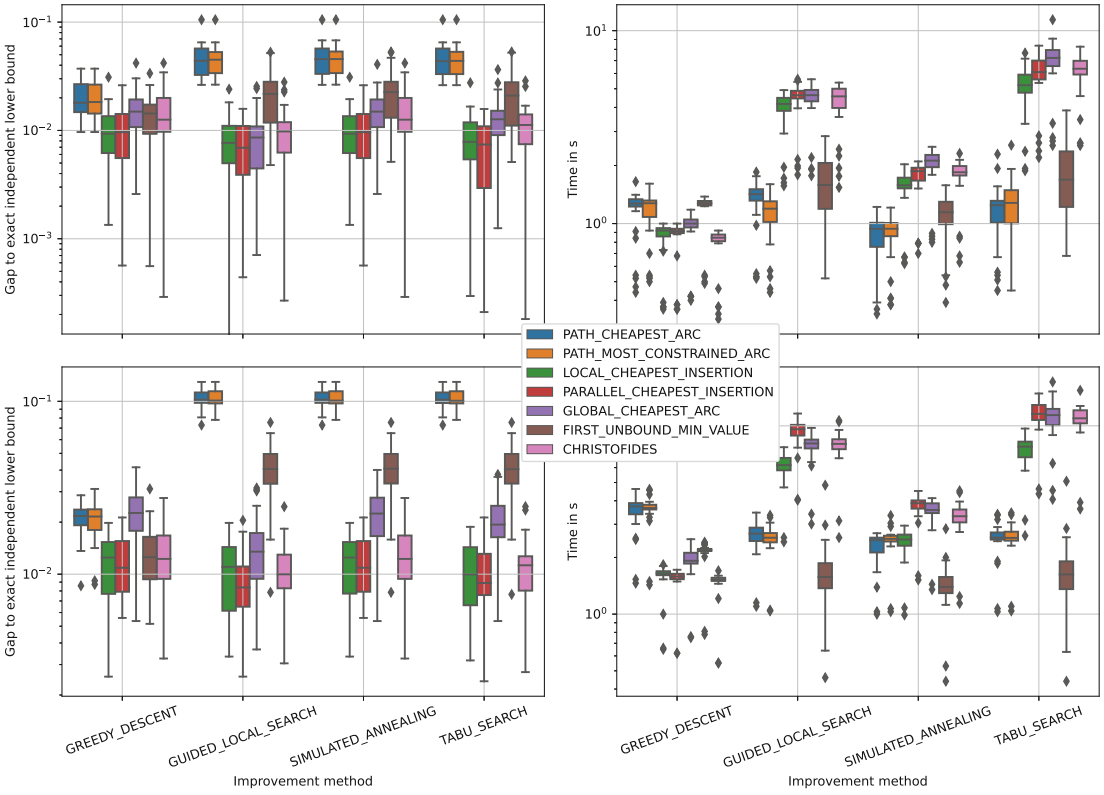


Figure 3.5: Comparison of gaps and running times for different construction heuristics and improvement methods, solving the root independent lower bounds using our Google OR-Tools HCVRPH model. Top left: comparison of gaps for instances with 18 teams. Top right: comparison of running times for instances with 18 teams. Bottom left: comparison of gaps for instances with 22 teams. Bottom right: comparison of running times for instances with 22 teams.

Table 3.2: Comparison of beam search with $\beta = 1\,000$ over different state ordering heuristics on 90 randomly generated test instances with 8, 10, and 12 teams, using rounded Euclidean distances, evenly split. Mean values of final solution lengths and standard deviations over 30 test instances are shown. As baseline, SHORT uses only the currently shortest path length to a state $g(s)$ as guidance in the beam search, and no costs-to-go estimate derived from the state ($b = 0$).

class	SHORT	CVRP	CVRPH	HCVRP
$\mathcal{I}_{L^2}^{8}$	34412 ± 5088	33034 ± 5109	32965 ± 5071	32919 ± 5051
$\mathcal{I}_{L^2}^{10}$	55019 ± 5872	51723 ± 5988	51269 ± 5808	51227 ± 5727
$\mathcal{I}_{L^2}^{12}$	79699 ± 7293	74231 ± 6933	73700 ± 6456	74722 ± 7250

Heuristic guidance. So far, we considered only heuristic solutions to the independent lower bound problem to see whether there the resulting gaps are plausible. This allows no direct judgement on whether this gives good guidance for the beam search, since also the variability of the heuristic quality over the layers of the search is deemed important. As first sanity check of whether HCVRP is suitable as guidance for solving the TTP

with our beam search approach, we evaluate its performance on artificially generated instances with smaller numbers of teams in $\{8, 10, 12\}$ from [57] in comparison with guidances by the exact CVRP and CVRPH bounds. Table 3.8 shows the mean and standard deviation of the objective values achieved by beam search with a beam width of 1 000 for different instances classes, each of size 30. HCVRP with the *local cheapest arc* construction heuristic and the *greedy descent* improvement heuristic performs similarly to its exact counterpart, the CVRP bound. With a Wilcoxon signed rank sum test on our sample with a significance level of $\alpha = 5\%$, we cannot reject the null hypothesis that they have equal performance—still, a possible type II error should be kept in mind.

In the next experiments, we tune algorithmic parameters on our larger artificial instances with 18, 20, and 22 teams, where the exact bound calculations become intractable and therefore heuristic solutions become more relevant. Based on the results from the root lower bounds before, we have fixed *greedy descent* as improvement method to calculate the bounds, where we observe a good trade-off between runtime and solution quality. As construction heuristic, we find that *parallel cheapest insertion* for HCVRP and *first unbound min value* for HCVRPH provide the best guidance. For HCVRPH, the other methods like *Christofides* or *parallel cheapest insertion* fall behind and seem to not work well with our Google OR-Tools problem formulation of the corresponding vehicle routing problems.

3.7 Beam Width, Symmetry Breaking, and Team Orderings

In this section, we study the impact of different aspects of the beam search on the solution quality. More concretely, we evaluate different combinations of beam width, team ordering, and reflective symmetry breaking on artificial random instances and a selected subset of benchmark instances from the literature to justify preparatory design choices for the subsequent computational study in Section 3.9.

Beam width. Intuitively, we seek to increase the beam width as much as possible within our computational and memory limits to explore a wider range of the solution space. Still, the monotonic increase in solution quality is not guaranteed since we might introduce misleading states. In our randomized multi-start beam search setting, this is to some extent mitigated, since we add noise to the guidance for diversification and therefore different parts of the solution space may be explored in each run even with the same beam width.

In Table 3.3 the number of feasible solutions found and the solution quality for two exemplary team sizes $n \in \{8, 20\}$ of our random Euclidean instances are compared for three orders of magnitude of the beam width $\beta \in \{1\,000, 10\,000, 100\,000\}$ and different configurations of team ordering and symmetry breaking; the latter two are described in the remaining paragraphs. Since the search may run into dead-ends due to a limited look-ahead capability of our approach in favor of construction speed and the constraint

Table 3.3: Number of feasible solutions found on random Euclidean instances with 30 instances for two exemplary team sizes $n \in \{8, 20\}$ and beam widths $\beta \in \{1\,000, 10\,000, 100\,000\}$ and different combinations of team orderings and reflective symmetry breaking, also random team ordering run once per instance. Due to dead-ends in the search, we observe that an increased beam width is required for increasing team sizes to find feasible solutions with high probability. Furthermore, the mean solution lengths $\bar{u}$ and standard deviation σ_u over the feasible results of all 12 configurations show the unsurprising tendency that higher beam widths lead to higher expected solution quality.

n	β	highest_total_distance			lexicographic			lowest_total_distance			random			#inf	$\bar{u}$	σ_u
		away	home	none	away	home	none	away	home	none	away	home	none			
8	1 000	30	28	29	30	30	30	30	30	30	30	30	29	4	32975.0	5023.2
	10 000	30	30	30	30	30	30	30	30	30	30	30	30	0	32401.4	4933.5
	100 000	30	30	30	30	30	30	30	30	30	30	30	30	0	32072.8	4863.3
20	1 000	27	30	29	25	29	28	27	28	28	29	29	28	23	201995.0	14756.8
	10 000	29	29	29	30	30	30	30	30	30	29	30	28	6	195383.6	14091.9
	100 000	30	30	29	30	30	30	30	30	30	30	30	30	1	191079.3	13656.5

satisfaction aspect of the problem itself, we see that high beam widths are required to find feasible solutions with high probability, in particular for a large n. In our previous work ([57], Table 3), we observe the same behavior for NL and CIRC benchmark instances comparing beam widths 1 000 and 10 000, where only with a beam width of 10 000 we could find feasible solutions for all instances—also better solutions are consistently found with the larger beam width. This should justify the use of a beam width $\geq 100\,000$ for the experiments in our computational study.

Reflective symmetry breaking. Symmetry breaking has been proven beneficial for exact methods for instance by Uthus et al. [154, 155]. It decreases the *runtime* by reducing the search space over which has to be reasoned. Since the beam search's runtime is already polynomially bounded, we seek to study the impact of reflective symmetry breaking along with different team orderings on the final *solution quality*. State transitions that are forbidden during construction by symmetry breaking should make space for other states to be explored further.

Due to the symmetric distances, schedules show a reflective symmetry. Look for instance at Fig. 3.1, where the solution displayed as a matrix can be mirrored across the horizontal middle line, i.e., the whole schedule could be reversed round-wise without changing the total sum of travel distances. To break this symmetry, we employ two different symmetry breaking methods adopted from Uthus et al. [155], there coined "symmetry-H" and "symmetry-A". We enforce for a given selected team—the pivot—that more away (home) games are scheduled in the first half of the schedule. This works because the number of away (home) games is odd. To respect this during construction, we additionally disallow home (away) games for the pivot at state s in the first half of the schedule, if playing it would make the accommodation of enough away (home) games in the first half impossible. Imagine an instance with eight teams and an abstract partial schedule $\mathcal{HH}$ ($\mathcal{AA}$) for the pivot, where two home (away) games have been played and five games are remaining in the first half of the schedule. Would it be continued by another home (away) game, then we could not play the minimum requirement of four away (home) games in the first half

Table 3.4: Relative objective values (in percent) to the instance-wise best objective of single beam search runs with beam width 10 000 for selected NL instances guided by the CVRP bound and a beam width of 100 000 guided by the HCVRP bound for selected NFL instances over different combinations of team orderings and reflective symmetry breaking, also random team ordering run once per instance. For one configuration and instance, no feasible solution has been found. A Friedman test cannot reject the hypothesis that all configurations perform equally well with $\alpha = 5\%$. The median relative objective values are for all configurations approximately in the range of $1 - 2\%$.

inst	highest_total_distance			lexicographic			lowest_total_distance			random		
	away	home	none	away	home	none	away	home	none	away	home	none
NL8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.7	0.4	0.0
NL10	1.9	0.8	0.8	2.1	2.4	2.1	2.2	0.0	0.0	1.1	0.9	0.0
NL12	1.9	2.8	2.0	0.0	0.6	0.0	2.4	3.0	2.3	3.2	0.8	1.9
NL14	0.8	4.3	0.8	2.4	2.2	2.4	5.0	4.5	5.0	0.0	3.9	3.6
NL16	2.6	0.0	0.0	0.8	1.8	1.8	5.0	1.9	1.9	1.5	3.1	3.4
NFL16	0.1	1.2	0.8	1.5	1.3	1.3	0.0	0.4	0.0	1.8	1.1	0.9
NFL18	3.5	0.9	0.9	2.5	1.1	1.1	0.0	1.2	0.0	1.7	0.6	0.3
NFL20	6.7	3.4	3.4	2.1	1.1	1.1	2.4	4.5	∞	1.2	0.0	4.5
NFL22	0.0	0.5	1.9	3.7	1.9	1.9	3.1	3.2	3.1	1.9	1.7	2.6
NFL24	5.9	4.3	5.2	1.7	0.0	2.6	2.0	2.3	1.2	1.6	3.0	2.7
median	1.9	1.1	0.9	1.9	1.2	1.5	2.3	2.1	1.6	1.6	1.0	2.2

of the schedule without violating the *at-most* constraint.

Team ordering. As static team ordering during the layer-by-layer construction of the beam search, we consider lexicographic, highest (lowest) total distance to other teams, and random ordering as control group. As suggested by Uthus et al. [155], the motivation for the sorting by total distance is to decide first for teams that are farther away (or closer) to the other teams, to observe earlier in the search an impact on the costs. For a beam search run, we have to select one out of these four team orderings and either no, *away*, or *home* reflective symmetry breaking, resulting in 12 different configurations.

Evaluation on benchmark instances. We evaluate the performance of the 12 different configurations on 10 different instances from the benchmark set of the literature (NL8-16, NFL16-24). For the NL we use the CVRP bound as guidance and a beam width of 10 000, for the larger NFL instances HCVRP as guidance and a higher beam width of 100 000. We show relative objective values in percent to the instance-wise best configuration in Table 3.4. Multiple different configurations yield the best solution for at least one instance, and the median relative objective values are approximately within the range of $1 - 2\%$ for all configurations. Given our sample, a Friedman test with significance level $\alpha = 5\%$ cannot reject the null hypothesis that all configurations have equal performance, while a possible type II error should be kept in mind. It could also be that higher beam widths overshadow the effects of the team ordering, but small beam widths are not interesting for us and therefore not considered here.

Since no configuration seems to set itself substantially apart from the others, we decide to keep the symmetries in all the experiments and use lexicographic sorting for single-run beam search and random team ordering for diversifying the search in randomized multi-start beam search runs. After the preparatory experiments of the last paragraphs and resulting design choices, we are about to move to the main computational study, where we compare HCVRP with HCVRPH guidance and discuss the impact of a local search performed after the beam search as postprocessing. We finally make a head-to-head comparison with a state-of-the-art approach—Traveling Tournament Simulated Annealing (TTSA) by Anagnostopoulos et al. [4]—which we reimplemented, and compare with the best-known results over a wide range of benchmark instances from the literature. Before that, we make a slight detour to weighted A* search for the TTP.

3.8 Weighted A* Search†

As discussed in the methodology in Section 2.2, A* search guarantees optimality given an admissible heuristic but generally suffers from exponential space complexity in the case of hard problems. One way to avoid this is iterative deepening A* [93] where the search graph is explored with an iteratively increasing cut on the f-value in a depth-first fashion, keeping the memory linearly bounded in the depth. Optimality and completeness are retained and the number of expanded nodes is asymptotically equal to A*. A sophisticated parallelized variant by Uthus et al. [155] allows solving TTP instances up to 10 teams within days. Still, a drawback in practice can be the numerous re-expansions of already visited states. We, therefore, study the performance of weighted A* [118], which trades optimality with bounded suboptimality in the hope of substantially reducing the number of expanded states. This is performed by rescaling the heuristic with a weight $w > 1$ so that it provides more pruning power and puts more focus on the nodes that look promising for the heuristic. As the heuristic may become inconsistent, it may become necessary to re-expand states to guarantee w-optimality, otherwise, the analysis is more intricate, as discussed in the review of weighted A* by Ebendt and Drechsler [48].

The high-level description of weighted A* for the TTP is displayed in Algorithm 3.2. It features state re-expansions by retaining a closed list and performing duplicate checks on expansion. Furthermore, to reduce the number of children generated, a cut u on the (unweighted) f-value can be supplied. As discussed by Wilt and Ruml [164], increasing w does not necessarily reduce the number of nodes expanded since the search can then get stuck in local extrema. Furthermore, for the TTP we may have dead-ends in the search, real ones by the constraints and artificial ones by the f-value cut and symmetry breaking. The algorithm is complete in the sense that if there is a solution satisfying the cut u, then it will be eventually found. Practical problems are likely the memory footprint and also runtime. In the computational study, we will study the performance of this approach over different parameter combinations for the weight w and the f-value cut u. If we do not have a cut, the focus is on finding a w-optimal solution, whereas when we have a cut, to prove either its w-optimality or even a lower bound, if no feasible solution is found.

Algorithm 3.2: Weighted A* Search for the TTP.

Input: number of teams n, distance matrix d, start state s_{start}, terminal state s_{t}, admissible heuristic b, beam width β, weight $1+\epsilon$, f-value cut u
Output: feasible schedule T with bounded suboptimality

priority queue $Q \leftarrow \{s_{\text{start}}\}$ sorted by weighted f-value;
closed list C as hash map with state as key, so far best f-value as value;
$s_{\text{final}} \leftarrow ()$;
while $Q \neq \emptyset$ **do**
  $s \leftarrow Q.pop$;
  break if state $s = s_{\text{t}}$ setting $s_{\text{final}} \leftarrow s_{\text{t}}$;
  continue if state $s \in C \wedge C[s] \leq f(s)$;
  $C[s] \leftarrow f(s)$;
  if *s has games to play* **then**
    $t \leftarrow$ next-team(s);
    foreach $(i,j) \in \{(i',j') \in P_t^s \mid r_{i'}^s = r_{j'}^s\}$ **do**
      $f(s') \leftarrow g(s') + (1+\varepsilon) \cdot b(s')$; `// by incremental evaluation`
      if *feasibility-and-symmetry-breaking-check*$(s,(i,j)) \wedge f(s') \leq u$ **then**
        $s' \leftarrow$ copy s and make transition by playing (i,j) and updating the state along with cached data accordingly;
        $s'.game\_played \leftarrow (i,j)$;
        $s'.parent \leftarrow s$;
        include s' into Q respecting weighted $f(s')$;
      end
    end
  else
    create going-home transitions for s to s_{t};
    include s_{t} into Q respecting weighted $f(s_{\text{t}})$;
end
if $s_{\text{final}} \neq ()$ **then**
  return $s_{\text{final}}.current\_schedule$;
else
  return *no feasible schedule found adhering to the f-value cut u*;

For A* search, a small memory footprint for the search nodes (state plus auxiliary information) is even more desirable due to its worst-case exponential space complexity. To achieve this, we construct state graphs for each team and encode a complete TTP composed of such single team states, stored as pointers. A team state is a 4-tuple consisting of the remaining opponents to play away, the number of remaining (abstract) home games, the current position, and the remaining games allowed on the current streak. The arc weights are the distances between the two incident states' positions.

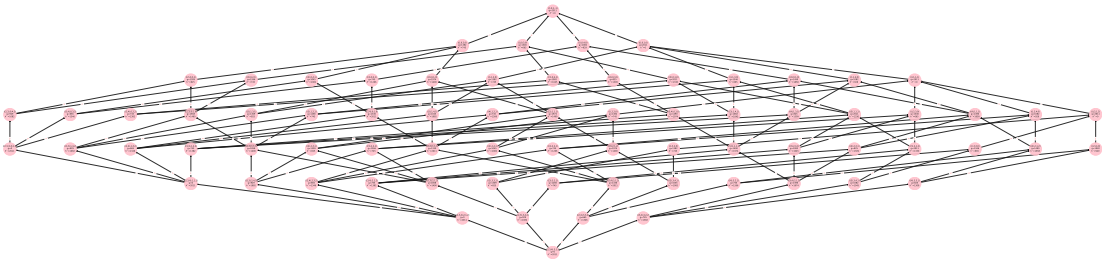


Figure 3.6: UFO-shaped state graph with abstract home games for instance NL4's first team, resulting in 74 nodes and 144 edges. In each node, the state, the shortest path g, and the perfect heuristic h^* are shown.

It corresponds to a directed acyclic graph which can be constructed by a breadth-first search—furthermore, the shortest paths g from the root to any other state, and from any state to the terminal node h^* (this amounts to a perfect single team heuristic) can be computed in linear time by corresponding forward/backward sweeps and dynamic programming. The graph without weights is the same for all instances with n teams and can be calculated once up-front, which grows exponentially in n. For a concrete instance, the forward and backward sweeps have to be performed to calculate the h^*, see Figure 3.6. Then we end up with a combined state representation and disjoint pattern database (as before, we use here the letter b to indicate that it is a bound and therefore an admissible heuristic) in a single data structure:

$$s = s^1 \oplus \cdots \oplus s^n \tag{3.16}$$

$$b(s) = h^*(s^1) + \cdots + h^*(s^n) \tag{3.17}$$

Since we explicitly model home games, the memory footprint and growth are substantially larger as compared to the approach from before used in the beam search, where we modeled only abstract home stands.

3.9 Computational Study: Beam Search

We present two computational studies of our randomized beam search approach, one based on a prototypical Python implementation [57] and the other on a Julia implementation [62], where we observe a substantial speedup. Additionally, we present results of a weighted A* search which provides bounded suboptimality.

3.9.1 Python Implementation

We conducted all our experiments on Intel Xeon E5-2640 processors with 2.40 GHz in single-threaded mode and a memory limit of 32 GB. We implemented our approach as a prototype in Python 3.7, being aware that an implementation in a compiled language would likely be substantially faster and have a smaller memory footprint. To solve the integer linear programs for the MNT bound, we use Gurobi 8.1.

Table 3.5: Memory demand for different lower bound lookup tables over the number of teams in GB assuming two bytes per bound value.

n	TSP	CVRP	CVRPH
14	0.003	0.009	0.127
16	0.016	0.047	0.75
18	0.079	0.237	4.27
20	0.390	1.172	23.43

Table 3.6: Runtimes in minutes for CVRPH bound calculations for NL14 to NL16 and CIRC14 to CIRC18.

	14	16	18
NLn	25	169	-
CIRCn	25	173	903

In Table 3.5, we see the memory demand for the three different lower bound lookup tables in GB assuming 2 bytes per bound value. Up to 16 teams, they all have reasonable size and our experiments have shown that the 16 teams instance bounds can be pre-calculated within three hours in a prototypic Python 3.7 implementation in our testing environment, see Table 3.6. 18 teams instances are also within reach with the strong CVRPH bound taking already 15 hours—we suppose that an order of magnitude in time can be saved using a compiled language. For larger instances, these numbers and the computation times increase dramatically, since the number of bounds grows for the CVRPH bound with $\mathcal{O}(n^3 2^n)$—already the TSP bound, being the weakest, needs 42 GB for 26 teams.

In Table 3.7, we present a comparison of the results obtained by the deterministic beam search with beam widths 1 000 and 10 000 for the instance sets NL and CIRC[9] [46] over the different lower bounds used in the state ordering. We see that the weak TSP bound does not provide good guidance and even misguides the search for larger instances, where using no bound ($b(s) = 0$) and sorting the nodes only by the currently shortest path length to them (SHORT) provides better results. Much better guidance can be observed for the CVRP-based bounds, where we see similar improvements over all instances.

Since the number of classic benchmark instances is limited and to further validate the guidance quality of the different bounds, we created two different types of random instances. First, the set $\mathcal{I}_{L^1}$, where we sample 30 instances for team numbers 8, 10, and 12 each on an integer grid of size 1000×1000 using Manhattan distances to compute the resulting distance matrices; second, the set $\mathcal{I}_{L^2}$, using the same sampling procedure but using the rounded to the nearest even integer Euclidean distances. This yields in total 180 additional test instances. We exclude the TSP bound from our further experiments since it did not show promising results for the tests on the NL and CIRC instances. In Table 3.8, we see mean values of final solution lengths and corresponding standard deviations when

[9] https://mat.tepper.cmu.edu/TOURN/

Table 3.7: Final solution lengths of deterministic beam search runs with different state ordering heuristics and beam widths with lexicographic team orderings. Sorting the states by the currently shortest path length to them (SHORT) does not use any lower bound; for a description of the TSP, CVRP, CVRPH, MNT lower bounds refer to Section 3.6. For CIRC16 with MNT we did not achieve a final result due to excessive runtime.

	$\beta = 1\,000$					$\beta = 10\,000$				
inst	SHORT	TSP	CVRP	CVRPH	MNT	SHORT	TSP	CVRP	CVRPH	MNT
NL6	24876	24759	23954	**23916**	**23916**	24876	**23978**	**23916**	**23916**	**23916**
NL8	42308	41977	40687	40687	40687	40970	41762	**39776**	**39776**	**39776**
NL10	67094	66469	62329	60713	62400	66087	64700	61129	60757	**60554**
NL12	131046	129209	116976	114499	114499	127238	119271	**113294**	114475	114824
NL14	217763	233765	211643	211116	211116	224537	219708	203519	**203279**	**203279**
NL16	309227	-	283985	285326	286085	301989	322567	276599	275562	**271251**
CIRC6	66	**64**	**64**	**64**	**64**	**64**	**64**	**64**	**64**	**64**
CIRC8	144	146	**134**	**134**	**134**	136	142	**134**	**134**	**134**
CIRC10	280	284	264	262	266	268	276	**246**	**246**	250
CIRC12	452	502	428	430	430	444	468	**418**	**418**	**418**
CIRC14	734	774	674	672	672	710	760	668	**656**	**656**
CIRC16	-	-	1012	1000	990	1012	1114	**956**	966	n/a

Table 3.8: Comparison of our beam search algorithm with $\beta = 1000$ over different state ordering heuristics on 180 randomly generated test instances with 8, 10, and 12 teams, using Manhattan and Euclidean distances, evenly split. Mean values of final solution lengths and standard deviations over 30 test instances are shown.

	$\beta = 1000$			
class	SHORT	CVRP	CVRPH	MNT
$\mathcal{I}^8_{L^1}$	42532 ± 5384	40530 ± 5214	40405 ± 5030	40405 ± 5030
$\mathcal{I}^{10}_{L^1}$	70049 ± 7280	65483 ± 6886	64760 ± 6689	64964 ± 6922
$\mathcal{I}^{12}_{L^1}$	99086 ± 7991	92838 ± 8089	91728 ± 7726	91465 ± 7694
$\mathcal{I}^8_{L^2}$	34412 ± 5088	33034 ± 5109	32965 ± 5071	32965 ± 5071
$\mathcal{I}^{10}_{L^2}$	55019 ± 5872	51723 ± 5988	51269 ± 5808	51057 ± 5829
$\mathcal{I}^{12}_{L^2}$	79699 ± 7293	74231 ± 6933	73700 ± 6456	73524 ± 6403

performing the deterministic beam search with the different state ordering heuristics on the randomly generated instances. The gap between SHORT and CVRP is well observable, especially with 10 and 12 teams. The gap between CVRP and CVRPH is closer, a Wilcoxon signed rank sum test reveals that CVRPH is significantly better than CVRPH with a significance level of $\alpha = 1\%$. The difference between CVRPH and MNT is for the L^1 distance instances inconclusive, and for the L^2 instances slightly in favor of MNT, but at the cost of substantially higher runtime due to the linear programs that need to be solved for every state. For further experiments, we, therefore, limit ourselves to the CVRP/CVRPH bounds.

Finally, Table 3.9 compares our randomized beam search variant with either lexicographic or random team ordering performed in parallel and independently on 30 cores with several

Table 3.9: Comparison of the final solution lengths of multi-start randomized beam search using either lexicographic team ordering or random team ordering (RTO) with 30 independent runs each, parameters $\sigma_{\text{rel}} = 0.001$, $\beta = 10^5$, and the CVRPH lower bound function (RBS-CVRPH) with the reported solution lengths of ant-colony optimization (AFC-TTP) [153], composite-neighborhood tabu search (CNTS) [39], simulated annealing (TTSA) [4], and population-based simulated annealing (PBSA) [160], where the latter is either used from scratch (PBSAFS) or starting from an already high-quality solution (PBSAHQ) provided by a TTSA run. †New best feasible solutions.

inst	RBS-CVRPH		RBS-CVRPH-RTO		AFC-TTP		CNTS		TTSA		PBSAFS		PBSAHQ	
	min	mean	min	mean	min	mean	min	mean	min	mean	min	mean	min	mean
NL12	112680	113594.6	112791	113581.5	112521	114427.4	113729	114880.6	112800	113853.0	**110729**	**112064.0**	n/a	n/a
NL14	192625	198912.6	196507	199894.8	195627	197656.6	194807	197284.2	190368	192931.9	**188728**	190704.6	**188728**	**188728.0**
NL16	266736	271367.1	265800	270925.9	280211	283637.4	275296	279465.8	267194	275015.9	**261687**	265482.1	262343	**264516.4**
CIRC12	410	415.7	410	414.6	430	436.0	438	440.4	n/a	n/a	**404**	418.2	408	**414.8**
CIRC14	632	641.0	**630**†	**640.7**	674	692.8	686	694.4	n/a	n/a	640	654.8	632	645.2
CIRC16	918	933.8	**910**†	931.6	1034	1039.6	1016	1030.0	n/a	n/a	958	971.8	916	**917.8**
CIRC18	1300	1322.0	1296	1320.4	1486	1494.8	1426	1440.8	n/a	n/a	1350	1371.6	**1294**	**1307.0**

state-of-the-art approaches on three difficult NL and CIRC instances. Each beam search run was conducted with beam width $\beta = 10^5$ and randomization parameter $\sigma_{\text{rel}} = 0.001$ resulting in equally gentle noise applied to the f-values of the states in every layer. The noise parameter was determined using irace [103] on the randomly generated instances. The table shows minimum and mean values for solution lengths of finally best solutions. We observe that we can compete well with the other mainly constructive approach "ant colony optimization with forward checking and conflict-directed backjumping" (AFC-TTP) from Uthus et al. [153] and the composite-neighborhood tabu search (CNTS) from Di Gaspero and Schaerf [39] on the NL instances and obtain better results than these for the CIRC instances, without hybridizing with a final local search. For CIRC instances we can also obtain similar results to population-based simulated annealing from scratch (PBSAFS) from Van Hentenryck and Vergados [160], which uses parallel simulated annealing. For the circular instances with 14 and 16 teams, we found new best feasible solutions, as of the time of writing according to Michael Trick's TTP web page. The strongest results overall for NL and CIRC are provided by simulated annealing (TTSA) from Anagnostopoulos et al. [4] and its parallel variant PBSA from Van Hentenryck and Vergados [160].

Runtimes of our approach are shown in Fig. 3.7 measured for deterministic beam search on the NL instances up to 16 teams for $\beta \in \{10^3, 10^4, 10^5\}$. For example, a run on an instance with 12 teams and a beam width of 10^5 takes roughly 10 hours. We believe it is possible to improve this further by an order of magnitude using a compiled language.

3.9.2 Julia Implementation

In our computational study using the Julia implementation, we conducted all our experiments on the same Intel Xeon E5-2640 processors with 2.40 GHz as before in single-threaded mode and a memory limit of 32 GB. We implemented our approach in Julia 1.6 interfacing Python 3.9 via PyCall to make use of Google OR-Tools 9.0, for which there are no native Julia bindings. We made the source code of our beam search solver

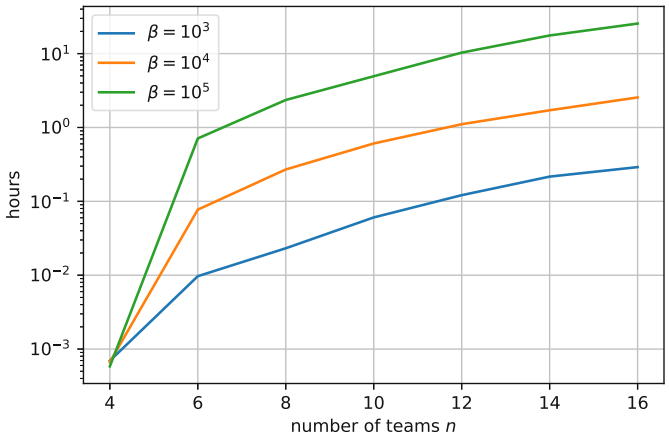


Figure 3.7: Runtimes in hours for deterministic beam search runs on NL instances with $\beta \in \{10^3, 10^4, 10^5\}$.

.

and all the considered instances available on GitHub.[10]

To make final algorithmic design choices, we first measure the impact on runtime and solution quality of the beam search utilizing different heuristic guidance methods, additional feasibility checks, and combining it with a final local search procedure. After that, we compare our approach with a reimplementation of the classical Traveling Tournament Simulated Annealing (TTSA) in Julia in a time-limited setting on the well-known NL benchmark set. Finally, we make a full comparison of our tuned randomized beam search with the best-known upper and lower bounds over a wide range of benchmark instances from the literature with up to 24 teams.

Guidance comparison. In Table 3.10 we compare the guidance quality by heuristically derived upper bounds to the CVRP bound and the CVRPH bound with and without dead teams check. Clearly, the guidance based on tighter bounds and detecting infeasible nodes in the beam search earlier results in better solutions to the TTP instances at the cost of moderately higher runtimes. In subsequent experiments, we, therefore, select HCVRPH with dead teams check to focus on solution quality.

Local search. We further study the impact of a final local search applied to the best-found solution of each beam search run. As neighborhood structure, we combine five classic neighborhood structures from Anagnostopoulos et al. [4] into one: Swap Homes, Swap Rounds, Swap Teams, Partial Swap Rounds, and Partial Swap Teams. We also allow to chain it with a second move but only from the swap neighborhoods, which lie in $\mathcal{O}(n^2)$, to keep the runtime small. The intermediate solutions on this chain may be infeasible but the final one has to be feasible. As discussed in Di Gaspero and Schaerf

[10] https://github.com/nfrohner/ttpbeam

Table 3.10: Comparison of mean relative upper bounds and mean runtimes in seconds of the HCVRP and the HCVRPH bound with (w/) and without (w/o) dead teams check (DTC) and with (w/) and without (w/o) applying local search (LS) on artificial test instances with 18, 20, and 22 teams, 30 instances each, and a beam width of 10 000. We observe that the mean relative upper bounds improve from left to right with a moderate increase in runtime. No randomization is used for the beam search guidance. The impact of the local search is quite small but leads to further improvement while its runtime only depends on the number of teams n and will not increase when larger beam widths β are used.

		HCVRP w/o DTC		HCVRP w/ DTC		HCVRPH w/o DTC		HCVRPH w/ DTC	
LS	n	$\bar{u}^{\text{rel}}$	$\bar{t}[s]$	$\bar{u}^{\text{rel}}$	$\bar{t}[s]$	$\bar{u}^{\text{rel}}$	$\bar{t}[s]$	$\bar{u}^{\text{rel}}$	$\bar{t}[s]$
w/o	18	0.140	663	0.135	804	0.130	838	**0.128**	961
	20	0.159	758	0.154	1034	0.144	836	**0.143**	1238
	22	0.181	950	0.174	1409	0.162	1223	**0.159**	1671
w/	18	0.137	+123	0.132	+104	0.128	+128	**0.125**	+113
	20	0.156	+248	0.151	+239	0.142	+234	**0.140**	+221
	22	0.177	+537	0.177	+483	0.160	+464	**0.157**	+479

[39], different moves may lead to the same neighbor, so we keep track of already-seen neighbors to avoid duplicate evaluation and chaining. As step function, we choose best improvement, either consisting of one or two moves in the neighborhood. We terminate when there is no more strictly improving neighbor. As seen in Table 3.10, this gives us a mild return improving the mean relative upper bounds for our concrete experiment by up to 0.3%. Since the runtime only depends on n and does not increase with the beam width β, we apply local search for our final experiments on the benchmark instances from the literature, where we increase the beam width by an order of magnitude.

Comparison with simulated annealing. For a fairer comparison with a state-of-the-art approach, we implemented the classic Traveling Tournament Simulated Annealing (TTSA) by Anagnostopoulos et al. [4] in Julia and reran experiments on the NL instances. The algorithm is local search based and uses the aforementioned five neighborhood structures to perform a guided random walk through the solution space of double round robin tournaments. During that, it permits constraint violations and uses strategic oscillation to traverse infeasible regions of the search space. A highly parallelized variant by Van Hentenryck and Vergados [160] is responsible for many best feasible solutions found so far over the benchmark instances.

After preliminary experiments, we made the following implementation detail choices, which are not specified in the original paper: We add a penalty proportional to the streak limit excess, to stronger penalize longer streaks. For example, a stand of five home games counts as two violations since it is two games longer than the maximum length of $U = 3$. Moreover, violations of the no-repeat constraint are counted team-wise, therefore we count a no-repeat constraint violation twice. Anagnostopoulos et al. [4] use five neighborhoods: Swap Homes, Swap Rounds, Swap Teams, Partial Swap Rounds, and Partial Swap Teams. In our implementation, each of those neighborhoods is equally likely

Table 3.11: Parameters used for our TTSA experiments with a time limit of two hours. The parameters for NL16 are taken from the fast cooling experiment of [4], the parameters for the other instances were changed accordingly based on the parameters of their full-run experiments. Initial temperatures (T_0), constraint violation penalty weights (w_0), weight change factors (δ and θ) and reheating factors (γ) stayed the same. Cooling factors (β) were set to 0.98, $maxC$ was halved, $maxP$ was divided by 100, and $maxR$ was multiplied by 200.

n	T_0	β	w_0	δ	θ	$maxC$	$maxP$	$maxR$	γ
8	400	0.98	4000	1.04	1.04	2500	70	2000	2
10	400	0.98	6000	1.04	1.04	2500	70	2000	2
12	600	0.98	10000	1.03	1.03	2000	14	10000	1.6
14	600	0.98	20000	1.03	1.03	2000	70	6000	1.8
16	700	0.98	60000	1.05	1.05	5000	70	10000	2

Table 3.12: Comparison of TTSA with fast-cooling and a time limit of two hours on the NL instances with a deterministic beam search run and multi-start randomized beam search runs with a beam width of 100 000 (except for NL8 with 500 000) running no longer than two hours with 30 runs for each instance. We observe that the randomization is effective in retrieving stronger best-found solutions and can compete well with TTSA(FC).

	Beam Search	Randomized Beam Search				TTSA(FC)				
instance	min	min	max	mean	std	min	max	mean	std	runs
NL8	39776	**39721**	39776	39741.2	27.0	**39721**	39721	**39721.0**	0.00	30
NL10	60034	59727	60167	**60052.3**	77.5	**59583**	60769	60113.9	346.0	30
NL12	113662	**112409**	115021	**113513.3**	2915.95	114920	130829	120651.4	2916.0	30
NL14	204305	**193891**	202777	**198590.9**	1363.86	195144	201506	198947.0	1363.9	30
NL16	275235	**265533**	274697	**270707.0**	1967.3	279226	287312	284137.1	2209.1	30

to be chosen when generating the next neighbor. We managed to obtain a moderate decrease in runtime using incremental evaluation for calculating changes of the objective value for the more round-local moves but found achieving this rather difficult, likely due to the rather small schedule size $n \times (2n - 2)$ and that we did not consider large n.

To generate an initial double round tournament (possibly with constraint violations), a randomized backtracking algorithm is used in the original paper, which constructs the schedule team-wise. It is observed that this does not scale well starting already from 16 teams with runtime outliers in the range of hours. As used in the ant colony optimization approach by Uthus et al. [153], we restart the construction in a Las Vegas algorithm fashion after a backtracking limit is hit, which we set empirically to 30 000. This allows the construction of random double round robin tournaments even up to 20 teams below one minute.

There are nine parameters for the algorithm that can be set: T_0 defines the initial temperature for the SA, β declares the cooling rate, w_0 is the initial weight with which constraint violations are penalized, δ and θ define how the weight changes when a new best feasible or a new best infeasible schedule is found. The number of iterations until equilibrium is reached is specified by $maxC$, the number of phases by $maxP$, and the number of reheats by $maxR$. While γ was not explicitly defined in the paper, we reasonably assumed it to be the reheating factor which is the factor by which the temperature is multiplied when reheating.

With our beam search approach, we believe we can construct competitive solutions for up

to 16 teams within two hours. We, therefore, compare it with TTSA using fast cooling, which is also described by Anagnostopoulos et al. [4], to quickly generate good solutions by a faster intensification of the search and more reheats. The used parameters are shown in Table 3.11, together with a fixed time limit of two hours. For beam search, we use the CVRPH bound for guidance, a beam width of 100 000 (except for NL8 with 500 000), a relative noise of $\sigma_{\text{rel}} = 0.1\%$, and a final local search as described before. We make also one run for each instance without noise. The beam search runs finish always under two hours. The results on the NL instances are displayed in Table 3.12, where we observe that our approach is at least competitive (leading also to stronger schedules for NL12-NL16) and that it is beneficial to use our type of stochastic ranking for diversification. Extensions of TTSA regarding refined neighborhoods [159, 99] and a much larger computational budget using parallelization [160], allow finding best solutions on a wide range of benchmark instances, as we will see in the subsequent comparison.

Table 3.13: Comparison with the best known lower bounds l^{best} and upper bounds u^{best} from the literature over the CIRC, GALAXY, NFL, and NL benchmark sets of our randomized beam search (30 runs, beam width 250 000) with random team ordering and Gaussian noise with a standard deviation of 0.1% of the root heuristic estimate, once guided by the exact CVRPH bound with resulting solution lengths u_1 and once by the heuristic version HCVRPH calculated via Google OR-Tools with resulting solution lengths u_2. Bold entries with a † indicate new best feasible solutions found.

instance	best l	best u	best u^{rel} [%]	min u_1	$\overline{u}_1$	min u_2	$\overline{u}_2$	min u_1^{rel} [%]	min u_2^{rel} [%]	Δ min u_1^{rel} [%]	Δ min u_2^{rel} [%]	$\bar{t}_1$ [h]	$\bar{t}_2$ [h]
CIRC12	388	404	4.1	406	413.2	408	414.5	4.6	5.2	0.5	1.0	4.6	5.4
CIRC14	588	630	7.1	630	637.4	634	643.3	7.1	7.8	0.0	0.7	8.0	9.1
CIRC16	846	910	7.6	912	926.3	934	946.9	7.8	10.4	0.2	2.8	12.2	13.5
CIRC18	1188	1294	8.9	1284	1309.9	1302	1324.2	8.1	9.6	†**-0.8**	0.7	22.2	22.5
CIRC20	1600	1732	8.3	-	-	1778	1803.0	-	11.1	-	2.9	-	31.3
GALAXY12	7034	7197	2.3	7251	7304.6	7180	7311.3	3.1	2.1	0.8	†**-0.2**	5.0	5.3
GALAXY14	10255	10918	6.5	11000	11096.9	10957	11125.3	7.3	6.8	0.8	0.4	7.7	9.1
GALAXY16	13619	14900	9.4	14655	14924.8	14820	14947.5	7.6	8.8	†**-1.8**	-0.6	12.9	13.1
GALAXY18	19050	20845	9.4	20489	20712.5	20646	20880.4	7.6	8.4	†**-1.9**	-1.0	24.1	22.5
GALAXY20	23738	26289	10.7	-	-	25818	26214.5	-	8.8	-	†**-2.0**	-	32.6
GALAXY22	31461	33901	7.8	-	-	35043	35335.7	-	11.4	-	3.6	-	46.6
GALAXY24	41287	44526	7.8	-	-	46127	46532.1	-	11.7	-	3.9	-	70.7
NFL16	223800	231483	3.4	238281	241921.2	238479	242232.1	6.5	6.6	3.0	3.1	13.2	13.3
NFL18	272834	282258	3.5	289250	294615.0	293914	298623.7	6.0	7.7	2.6	4.3	22.3	22.7
NFL20	316721	332041	4.8	-	-	341602	349917.0	-	7.9	-	3.0	-	34.0
NFL22	378813	402534	6.3	-	-	411241	419778.8	-	8.6	-	2.3	-	50.1
NFL24	431226	463657	7.5	-	-	469538	477533.1	-	8.9	-	1.4	-	76.4
NL12	108629	110729	1.9	112048	113107.8	111987	113309.8	3.1	3.1	1.2	1.2	5.0	5.3
NL14	183354	188728	2.9	193513	198575.9	194424	199230.3	5.5	6.0	2.6	3.1	7.7	9.0
NL16	249477	261687	4.9	267684	270172.3	267992	271786.6	7.3	7.4	2.4	2.5	13.6	14.7

Final comparisons. In Table 3.13, we finally compare the results of our randomized beam search approach with the best-known feasible solution values u^{best} from the literature. For the considered NL, CIRC, and NFL instances, these are from Van Hentenryck and Vergados [160], except for CIRC14 and CIRC16 which we could improve in our previous work. For GALAXY12, GALAXY14, GALAXY16, and GALAXY20 they are from Langford [99], for GALAXY18 and GALAXY24 taken from Hirano, Abe, and Imahori[11], and for GALAXY22 from Goerigk et al. [74]. The best-known lower bounds l^{best} were retrieved from the RobinX repository.[12] We performed 30 runs in parallel per instance with a beam width of 250 000 guided by either the exact CVRPH bound from 12 up to 18 teams or the HCVRPH variant from 12 up to 24 teams on the NL, CIRC, and NFL benchmark instances from [46] and the GALAXY benchmark instances from Uthus et al. [155]. For the HCVRPH bound, we use as construction heuristic *first unbound min value* together with *greedy descent*, since we find *local cheapest arc* unable to construct initial solutions for larger instances in a reasonable time when we add constraints for the minimum and maximum number of vehicles, which is necessary to tighten the bounds. Diversification is achieved by randomly shuffling the team ordering for each run and using Gaussian noise for the guidance values with a standard deviation of 0.1% of the root heuristic estimate, as tuned in our previous work.

We index performance figures of the runs guided by the exact CVRPH with 1 and by the HCVRPH with 2. The gaps $u_i^{\text{rel}}, i \in \{1, 2\}$ are calculated as $u_i^{\text{rel}}/l^{\text{best}} - 1$, where $\Delta \min u_i^{\text{rel}}$ are the relative gap differences between the best found u_i and the best known feasible solutions u^{best}. Values $\bar{t}_i$ are the mean runtimes in hours, excluding the bound precalculations for the exact guidance, which is once 70 minutes for the instances with 18 teams and otherwise less than 10 minutes. We observe a mean relative gap difference between the minimum over our 60 runs (30 with CVRPH, 30 with HCVRPH) and the best feasible solutions from the literature over the considered 23 instances of 1.2%. Furthermore, we report new best feasible solutions for five instances, one from CIRC and four from GALAXY.

3.10 Computational Study: Weighted A* Search†

Apart from beam search, we also implemented and tested weighted A* search in Julia 1.7. The main difference to the beam search implementation is that we enumerate the team graphs (see Section 3.8 and also [88]) with explicit abstract home games instead of only the home stands and use an array of pointers to teams states to represent a complete TTP state. Since A* is memory hungry, we performed first experiments on an AMD EPYC 7642 in single-threaded mode with 512 GB uniform memory and subsequent anytime weighted A* experiments on an Intel Xeon E5-2680 v4 with 768 GB non-uniform memory.

[11] no publication found, from RobinX repository

[12] `https://www.sportscheduling.ugent.be/RobinX`

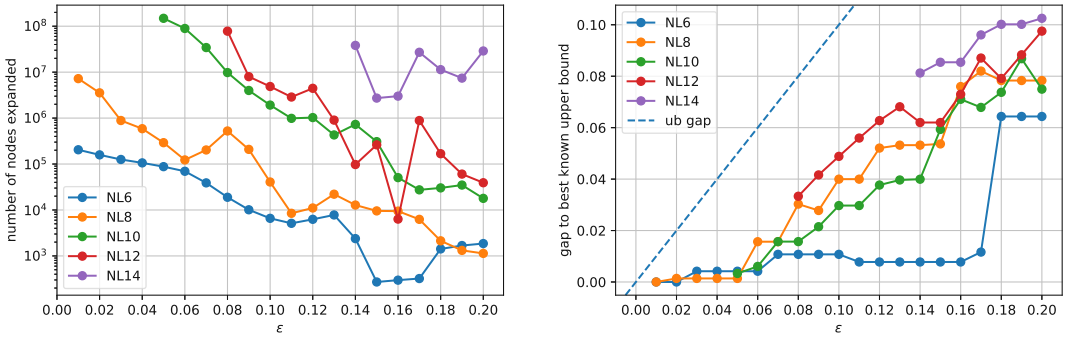


Figure 3.8: Results regarding numbers of expanded nodes and gaps to the best known feasible solutions for weighted A* runs with duplicate states detection on NL instances from 6 up to 14 teams over $\varepsilon \in \{0.01, 0.02, \ldots, 0.2\}$. Missing points indicate that we ran out of memory since we observe a non-monotonous exponential trend in expanded nodes when decreasing the weight parameter ε. The dashed line is an upper bound to the gap due to the suboptimality guarantees of weighted A*.

Implementation details. We check for duplicate states when dequeuing a search node from the open list. If we have found a shorter path to an already discovered state, we re-expand it, if the path is at least as long, then we safely discard the corresponding search node. All the reached search nodes and pointers to their parents are kept in memory to reconstruct the schedules and for the duplicate checks. These reached nodes are stored and preallocated in a contiguous static memory region with a size limit to reduce memory management activity and garbage collection effort. We use symmetry breaking on the away teams using the first team as pivot and static lexicographic team ordering as described in Section 3.7.

First results. The first results are shown in the plots in Figure 3.8 tested on the famous NL instances from 6 to 14 teams. On the left, we see the number of nodes expanded over varying weight parameter $\varepsilon \in \{0.01, 0.02, \ldots, 0.2\}$ providing a guarantee on the optimality gap. As expected, the trend when decreasing ε is an exponential increase in expanded nodes, although with substantial variability. For instances with more teams, the memory limit is hit earlier (for NL10 $\varepsilon_{\min} = 0.05$, for NL12 $\varepsilon_{\min} = 0.08$, and NL14, $\varepsilon_{\min} = 0.14$), since the node density grows, the number of nodes per f-value. On the right, we show the relative gaps to the best-known upper bounds. The tendency is that a tighter optimality guarantee results in better solutions but not monotonously, see the decrease for, e.g., NL12 from $\varepsilon = 0.13$ to $\varepsilon = 0.14$. For instances NL6, NL8, and NL10, the optimum is known and we observe an optimality gap below 1% already with $\varepsilon = 0.05$.

Dynamic team ordering. To further decrease the number of expanded nodes, we propose a dynamic team ordering mechanism. At every search node, we iterate through all teams that have not played in the current round and calculate for each the minimum increase in the f-value over their allowed games. Then we select as the next pivot team the one with the maximum over its minimum increases. The intuition is to tighten the bound and to detect inevitable cost increases earlier in the search tree. Furthermore, we

Table 3.14: Comparison weighted A* runs on selected instance/ε combinations plain, with duplicate detection (dup), with dynamic team ordering (dyn), and both (dd), where r^*_{exp} are the fractions of expanded nodes compared to the plain run and t^* the corresponding runtimes.

inst	ε	#exp	#succ	t [s]	$r^{\text{dup}}_{\text{exp}}$ [%]	t^{dup} [s]	$r^{\text{dyn}}_{\text{exp}}$ [%]	t^{dyn} [s]	$r^{\text{dd}}_{\text{exp}}$ [%]	t^{dd} [s]
NL6	0.01	214 382	496 187	2.1	95.4	2.3	61.8	**2.0**	**59.3**	2.2
NL8	0.04	755 368	2 653 367	3.8	77.8	4.3	50.6	**3.3**	**37.1**	3.4
NL10	0.08	70 337 826	222 503 742	242.6	13.9	106.0	21.9	339.9	**2.3**	**18.5**
NL12	0.09	10 326 694	37 283 179	39.5	77.3	57.6	3.9	3.6	**2.6**	**3.5**
NL14	0.16	3 017 268	30 834 322	27.6	99.6	61.8	26.6	**23.2**	**24.0**	24.1

check whether there is one team without any playable games. Then we know that this state has no feasible completions and can safely discard the node.

We empirically study the impact of duplicate detection and the dynamic ordering on the search by comparing the combinations (either one or both) with the baseline without duplicate detection and static team ordering, to see whether the additional overhead to the node expansions pays off. In Table 3.14 the number of expanded nodes and runtimes on selected instance/ε combinations are shown. We observe that both algorithmic features used together not only substantially reduce the search effort in terms of the number of expanded nodes up to two orders of magnitude, but also decrease the runtime by one order of magnitude. Another advantage is that the memory demand is reduced. We, therefore, use both for the remaining experiments.

Heuristics comparison. Other important aspects are the heuristics and symmetry breaking. Using the abstract home game formulation for the teams' state graphs, we automatically receive the CVRPH heuristic to guide the search. The JCVRPH heuristic provides further strengthening, which we can also calculate in reasonable time using directly the teams' state graphs—for details on the lower bound based heuristics, see Section 3.6. A weakening is the CVRP heuristic which does not include information about the home games. In Table 3.15, we compare the impact of different combinations of symmetry breaking and these heuristics on an exact A* search ($\varepsilon = 0$) for the instances NL6 to NL14 with a memory limit of 10^8 reached nodes. As expected, the strongest bound JCVRPH together with symmetric breaking proves optimality with either the smallest number of expanded nodes or provides the tightest lower bound, when the memory limit is hit. Further strengthening using a computationally efficient JCVRPH2+MNT bound would be interesting future work.

Memory efficiency. Since the successor creation rate is in the order of 10^5 nodes per second, we encounter earlier a memory problem than with runtime. Therefore we shall now focus on memory efficiency of the search by sensibly trading it for extra runtime. We reduce the size of a search node to only include the changed states of the teams that just played a game. This adds overhead to recover a state when dequeuing a node from the open list, where we have to traverse over a bounded number of predecessors.

Table 3.15: Comparison of best found f-value and number of expanded nodes of A* runs ($\varepsilon = 0$) using the best known feasible solution as cut and a limit on the number of reached nodes of 10^8 for different combinations of heuristics (heur) and symmetry breaking (sym). For NL6/NL8 proven optimal solutions were found and the number of expanded nodes is used as figure of merit, whereas for the others the best found f-values, i.e., lower bounds achieved within the memory limit are used.

	f-value					#exp				
sym	none	away		none		none	away		none	
heur	CVRP	CVRPH	JCVRPH	CVRPH	JCVRPH	CVRP	CVRPH	JCVRPH	CVRPH	JCVRPH
inst										
NL6	23916	23916	23916	23916	23916	384 403	161 921	**131 061**	304 775	232 942
NL8	39721	39721	39721	39721	39721	23 271 844	8 281 457	**6 239 131**	22 886 865	16 251 829
NL10	57525	57616	**57676**	57540	57603	15 941 328	16 393 288	15 905 523	16 005 183	15 495 812
NL12	107753	107788	**107826**	107787	107825	12 689 630	12 098 849	12 093 831	12 060 218	12 063 342
NL14	182818	182836	**182854**	182818	182824	7 384 490	7 591 273	7 168 704	7 384 490	7 064 109

For a memory-efficient duplicate detection, we use an open hash table in form of a fixed-length array with linear probing. As length, we take the next prime number greater than the maximum number of search nodes. We artificially limit the number of additional probing steps—extra steps when the slot determined by the hash function is occupied by a hash collider—to 10, to avoid long probing sequences. Preliminary experiments have shown that this limit is rarely hit. Including the size of the hash table and open list, this results in a constant search node size of 64 bytes, *independent* of the number of teams n.

Anytime search. We embed this memory-efficient weighted A* in an anytime A* search variant, where the optimality guarantee ϵ is iteratively decreased according to a given schedule, known from anytime repairing A* (ARA*) [102]. The so-far best feasible solution decreased by one (to enforce finding a strictly better solution) is used as primal cut to further reduce the growth of the open list through pruning. If in the current iteration given an ε and a primal cut u no goal state is found abiding the current cut, then we know that the corresponding solution is optimal. If a solution with length u' is found, we gain a corresponding lower bound by $u'/(1+\varepsilon)$, and maintain the maximum so far. If the memory limit is hit, no conclusion is drawn.

In Figure 3.9 we show the number of expanded nodes over ε and anytime plots regarding the solution quality as gap to the best-known solution on the instances NL6 to NL14, with a maximum number of reached nodes of 10^{10}. As heuristic, we use the JCVRPH bound together with away symmetry breaking. It can prove optimality for NL6 and NL8 quickly, find the optimal solution for NL10 (although not prove it), and provide competitive solutions to NL12/14 compared with the previous beam search runs, albeit with a larger memory footprint.

3.11 Conclusions and Future Work

We first investigated a beam search approach for the well-studied traveling tournament problem. To this end, we proposed a state space formulation, which is traversed by a restricted breadth first search to create heuristic solutions. This beam search is realized

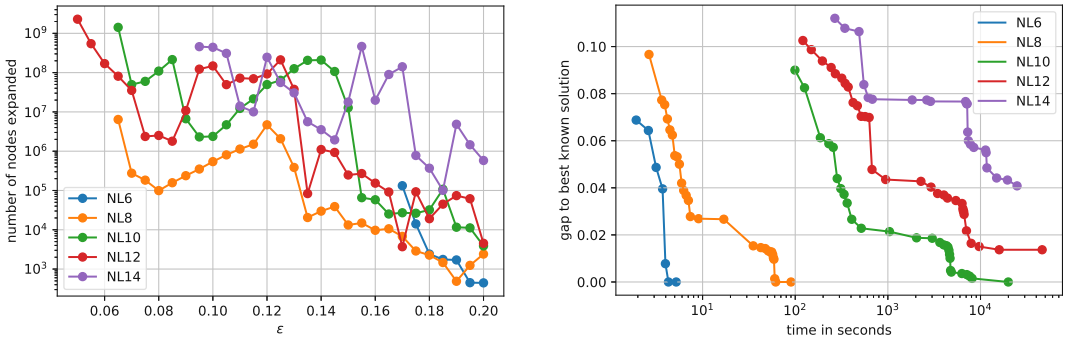


Figure 3.9: Anytime weighted A* results, left the number of of expandes nodes over ε, right the gaps to the best known solutions.

in a memory-efficient variant allowing for large beam widths. For guiding the search, we studied different lower bounds derivable from a state and also considered calculating these bounds inexactly but quickly, by solving related subproblems heuristically. Moreover, we introduced a randomized beam search variant that shuffles the team ordering and applies parameterized Gaussian noise to the guidance values in order to diversify the search when performing multiple runs in parallel.

We solved the capacitated vehicle routing problem instances for the lower bounds exactly, modeled as recursive constrained shortest path problems on a separate state graph for each team, and heuristically on-the-fly with Google OR-Tools. We found that with the latter, sufficient guidance quality can be retained to construct high-quality solutions for instances up to 24 teams, where the exact bound calculation is limited to instances with up to 18 teams.

We report a mean relative gap of 1.2% to the best-known solution values over 23 benchmark instances and new best feasible solutions for five instances, one from the CIRC and four from the GALAXY instance set, three constructed by guidance with exact bounds and two by heuristic estimates. The artificial instances created for tuning algorithmic parameters and the source code of our approach which we implemented in Julia were made available on GitHub. We compare our approach with our own implementation of a state-of-the-art simulated annealing approach in a time-limited setting of two hours and found our results to be mostly superior on the NL instances. While we could improve best known feasible solutions of difficult instances from the literature, the state-of-the-art approach for up to 24 teams still seems to be simulated annealing based approaches without time bounds and parallelization, which found the best-known solutions for all the considered NL and NFL instances, many CIRC and some GALAXY instance. Hybridizing our approach with simulated annealing could therefore be interesting future work.

The random team ordering and the Gaussian noise added to the guidance were an effective way to diversify the search when performing multiple runs. Promising future work includes the consideration of machine learning and more specifically transfer learning techniques on top of the proposed methods to possibly come up with even better guidance

functions. As seen for other approaches due to the complexity of the problem, we will investigate a heavily parallelized beam search variant in the next chapter.

We also studied a weighted A* search on our state space, which has the advantage of providing a suboptimality guarantee and being complete, i.e., returning a solution if there is one, which is not necessarily the case for beam search. We devised a memory-efficient variant using a stronger heuristic, dynamic team ordering, and duplicate detection to substantially reduce the number of expanded nodes required. This allows proving optimality quickly for NL6 and NL8, and find the optimal solution for NL10 (which we did not achieve with beam search), and high-quality results for NL12/14, with a memory limit of 10^{10} reached search nodes. Still, the memory demand at some point becomes a problem. As shown by Uthus et al. to solve instances with ten teams, we believe a parallelized variant using subtree splitting to distribute subproblems on different workers with large amounts of memory and many-core parallelization, possibly with GPUs, to speed up the local solving process is necessary to attack NL12 instances, together with a further strengthening of the lower bounds.

CHAPTER 4

Parallel Beam Search for Combinatorial Optimization

Inspired by the recent success of parallelized exact methods to solve difficult scheduling problems [72, 71] and successful parallelization efforts for breadth-first search [166], we present a general, parallel beam search framework for combinatorial optimization problems. Beam search is a heuristic search algorithm traversing a search tree layer by layer while keeping in each layer a bounded number of promising nodes to consider many partial solutions in parallel, thereby already suggesting a cooperative handling of the workload. We propose a variant that is suitable for intra-node parallelization by multithreading with data parallelism based on layer synchronization. Diversification and inter-node parallelization are combined by performing multiple randomized runs on independent workers communicating via the message passing interface (MPI).

For sufficiently large problem instances and beam widths, our prototypical implementation in the JIT-compiled Julia language admits speedups between 30–42× on 46 cores with uniform memory access for three difficult classical problems, namely permutation flow shop scheduling (PFSP) with flowtime objective, the traveling tournament problem (TTP), and the maximum independent set problem (MISP). This allows us to perform large-scale runs with beam widths in the order of millions to find 11 new best feasible solutions for 22 difficult TTP benchmark instances with up to 20 teams with an average wallclock runtime of about one hour per instance.

An extended abstract for the parallel beam search framework was first published in the proceedings of the SoCS 2022 conference and presented in the respective poster session:

> Nikolaus Frohner, Jan Gmys, Nouredine Melab, Günther R Raidl, and El-Ghazali Talbi. Parallel beam search for combinatorial optimization (extended abstract). *International Symposium on Combinatorial Search*, 15(1):273–275, 2022

This chapter is mostly based on the successive full paper presented at the 51st International Conference on Parallel Processing (ICPP) in the Parallel and Distributed Algorithms for Decision Sciences (PDADS) workshop and published in the respective proceedings:

> Nikolaus Frohner, Jan Gmys, Nouredine Melab, Günther R Raidl, and El-Ghazali Talbi. Parallel beam search for combinatorial optimization. In *51th International Conference on Parallel Processing Workshop*, ICPP Workshops '22. Association for Computing Machinery, 2022

Here, this work is further extended by a study of parallel beam search for the MISP and we further elaborate on related work regarding beam search and parallel breadth-first search. Moreover, we additionally consider an anytime variant with iterative widening of the framework.

4.1 Introduction

Over the last decade, the increase in computational power can be largely attributed to parallelization, while single-threaded performance tends to saturate, as indicated by Figure 2.6. Clusters with hundreds of nodes equipped with multiple CPUs and GPUs consisting of thousands of cores become available to more and more researchers all over the world. With semiconductor geometries still shrinking we can expect the core density to still grow even further over the next years.

Accordingly, solution approaches to difficult combinatorial optimization problems were designed and implemented to benefit from these technological transitions. One solution paradigm is to formulate a problem recursively and model the solution process as traversal of a search tree, for instance as done by classic branch-and-bound (B&B) algorithms or A* search. In this work, we describe a parallel beam search framework for combinatorial optimization problems. Beam search is a well-known search method where a search graph is traversed layer-wise keeping a bounded number of nodes in each layer, resulting in a polynomial number of nodes to be evaluated—a truncated breadth-first-search (BFS).

Beam search has been used to construct high-quality feasible solutions in the context of branch-and-bound, standalone, or combined in a hybrid setting with an improvement heuristic like local search. Quite recently, strong results have been obtained on difficult scheduling problems, see Libralesso et al. [101] on Permutation Flow Shop Scheduling (PFSP) and Frohner et al. [57] on the Traveling Tournament Problem (TTP). The crucial parts are always the evaluation of the nodes, the *guidance*, and the *beam width*, limiting the maximum width of the search tree.

We propose a framework for combinatorial optimization problems that admit a recursive formulation, i.e., where there exists the notion of partial solutions with a corresponding state which we can evaluate to guide our search. While at first focusing on CPU execution models, we already have heap-less systems with preallocated memory regions in mind

to pave the way for GPU implementations. Parallelization is nested in two levels, the inter-node and intra-node level. The latter is achieved by evaluating nodes, selecting the most promising nodes, and making the state transitions by multithreading with data parallelism on shared memory while minimizing sequential code part runtimes. On the inter-node level, multiple randomized runs are distributed on independent workers to diversify the search.

We implement the framework in the Julia programming language [16], available on GitHub,[1] and study it for three well-known $\mathcal{NP}$-hard optimization problems, namely Permutation Flow Shop Scheduling (PFSP) with flowtime objective, the Traveling Tournament Problem (TTP), and the Maximum Independent Set Problem (MISP), based on previously proposed search spaces and guidances of state-of-the-art approaches from the literature. We observe an intra-node parallel efficiency with 46 cores of around 60–90% on sufficiently large problem instances and beam widths, resulting in speedups between 30–42×. Combining large beam widths in the millions and diversification, we could derive 11 new best feasible solutions for 22 difficult TTP benchmark instances with an average wallclock runtime of about one hour per instance and a mean gap to the best-known solutions of virtually 0%, matching the state of the art.

The next section discusses previous works on tree exploration approaches related to our test cases. Afterwards, we give a brief formalization of combinatorial optimization problems suitable for our state-space search setting. Next, we describe the details of our parallel beam search approach and the concrete implementations for our three example problems. A computational study with focus on parallelization aspects is presented afterwards. Finally, we conclude and discuss promising future work.

4.2 Related Work

Beam search was invented in the 70s in the AI community for the speech recognition system HARPY described in the PhD thesis of Lowerre [105] supervised by R. Reddy. The challenge was to combine advantages of two previous systems, one using a best-first search approach and the other a full breadth-first search through a state transition network. The compromise here was to construct a number of solutions in parallel guided by strong heuristics until acceptance, avoiding the backtracking of best-first search and reducing the search complexity of a breadth-first search.

Ow and Morton [113] provide a short historical discussion and a general introduction, related algorithmic design decisions, trade-offs, and empirical properties of beam search. They introduce *filtered* beam search, where in a first step a fast, local evaluation is used to reduce the set of successors down to a *filter width*. These are then further evaluated in a second step by a global evaluation function (one that seeks to estimate the least costs to a goal node) and only the best up to a *beam width* are kept. They provide an empirical study of filtered beam search applied to two example scheduling problems.

[1] `https://github.com/nfrohner/parbeam`

Bisiani [17] and Zhang [165] formulate beam search more general in terms of pruning rules added to any underlying search algorithm like breadth- or depth-first search to reduce search complexity. If not explicitly stated otherwise, we view beam search throughout this work as a layer-wise state-space traversal where the up to the beam width best nodes according to a heuristic are pursued further while the excess nodes in each layer are irreversibly pruned.

Zhang [165] emphasizes the incompleteness of beam search, i.e., there is no guarantee that we find a solution if there is one, and proposes an *complete anytime* beam search by iterative weakening of the pruning rules, until an acceptable solution has been found or until the last iteration, where no inadmissible pruning rule has been applied.

Korf et al. [96] in the context of A* search and Zhou and Hansen [168] in the context of BFS, coined breadth-first heuristic search, introduce frontier state space search methods, where only the search frontier (the open list) and nodes close to it are kept. This allows being memory efficient since no closed list is used. The solution reconstruction is performed by a divide-and-conquer method, keeping nodes in an intermediate layer and solving subproblems (from start to intermediate node of best-found solution plus from intermediate to goal node) recursively. For memory efficiency, we also keep only the current and the next layer in memory, but directly encode the solution into the search node. A large part of a search node's memory footprint is associated with the state and caching information to facilitate fast incremental evaluation of successors.

Zhang and Hansen [166] propose a parallel breadth-first heuristic search on shared-memory architectures and demonstrate high parallel efficiency with six threads on a NUMA system for the 15-puzzle problem. Workload distribution is performed layer-wise, introduced as *layer synchronization*, with dynamic scheduling of chunks of nodes and concurrent hashing for layer-wise duplicate detection. In a project work, Guan presents a layer synchronization based parallel beam search algorithm for a 3D matching problem.[2] We adopt and suitably extend this concept for our large-scale beam search, additionally parallelizing the selection of the best nodes, with the focus on fast implementations for different combinatorial optimization problems.

Hifi and Saadi [81] present a controller-worker approach using a global list to guide beam search runs over subtrees on a cutting-stock problem, where we restrict ourselves with straightforward randomized multi-starts and possible pruning when applying the beam search iteratively.

Gmys et al. [72] present a highly efficient parallel branch-and-bound approach for the PFSP with makespan objective, able to improve best-known solutions of many benchmark instances including numerous new optimality proofs. In related work, Gmys [71] develops a sophisticated GPU-accelerated branch-and-bound framework with which long-standing Taillard PFSP benchmark instances [143] were solved to optimality. In it, the beam search algorithm as proposed by Libralesso et al. [101] is applied to obtain primal bounds, which

[2] https://isaacguan.github.io/projects/comp5704

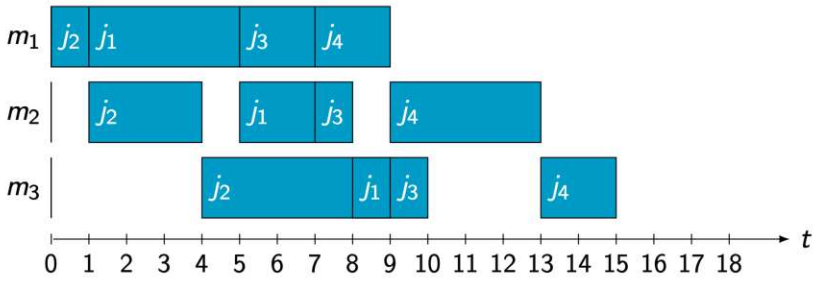


Figure 4.1: Example schedule of PFSP instance with $n = 4$ jobs and $m = 3$ machines. The resulting sum of finishing times is 42 and the makespan is 15.

combines forward and backward branching and a guidance function based on a weighted sum of idle time and a fast lower bound to achieve state-of-the-art heuristic results for the makespan optimization of the PFSP. The latter algorithm is also competitive for the flowtime optimization case and serves as basis for our parallel PFSP beam search implementation. Its original implementation is based on the Discrete Optimization Global Search (DOGS)[3] framework, an anytime tree search framework written in Rust, related to which we did only find computational results in single-threaded mode in the literature.

The traveling tournament problem was introduced by Easton et al. [46] and has since then been a prominent benchmark problem for all kinds of metaheuristics and exact approaches due to its nature of having a rapidly exploding but highly constrained search space. An iterative-deepening A* approach from Uthus et al. [155] is the state-of-the-art exact method to tackle it. This method allows solving instances with 10 teams in between one day and one week using 120 workers in parallel with subtree splitting. Improved lower bounds and heuristic solutions could be found for instances with up to 14 teams. Inspired by this successful tree search approach, Frohner et al. [57] present a randomized beam search algorithm that allowed finding new best feasible solutions for two benchmark instances up to 18 teams, on which we base the implementation of the TTP parallel beam search solver in our framework.

We base and modify our formulation for the MISP on the works of Bergman et al. [12, 13] and our works [53, 54] where decision diagrams, close relatives of state graphs, are studied to construct discrete relaxations for MISP instances.

4.3 Formalization

We build on the state-space formulation for combinatorial optimization problems as presented in Section 2.2. To reiterate, assume that every solution $x \in S$ is represented by at least one sequence of assignments of values from an arbitrary finite domain D to n decision variables $x_l \in D$, $l = 1, \ldots, n$. All possible assignments are described by a directed acyclic graph $G(C) = (V, A)$, the *search graph* belonging to the problem instance

[3] https://github.com/librallu/dogs

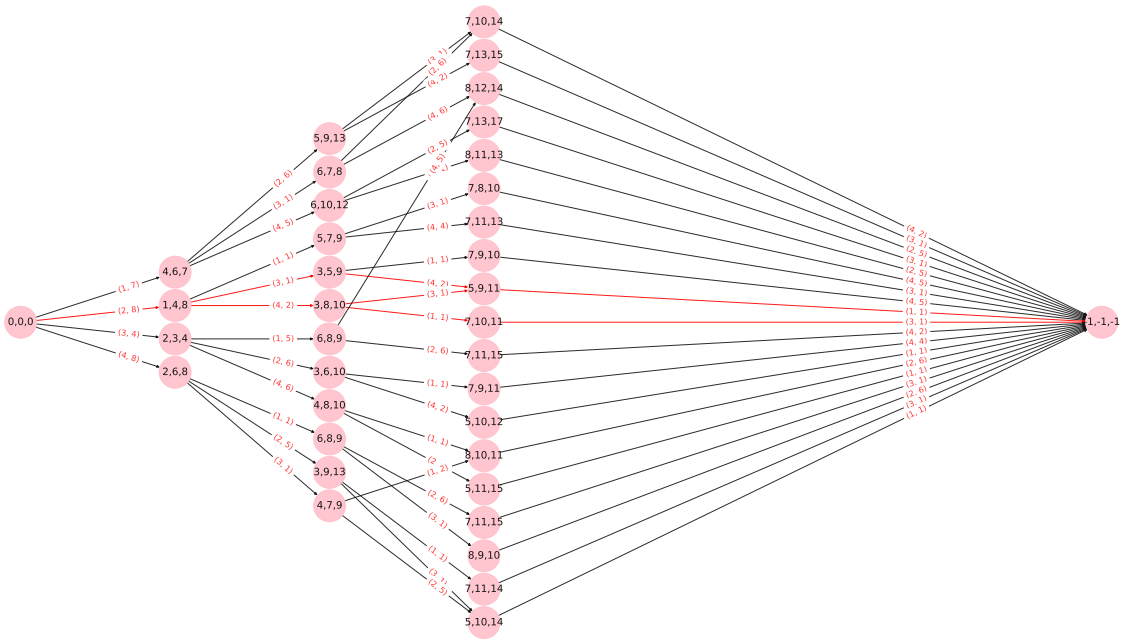


Figure 4.2: State graph of a PFSP example instance with makespan objective. Optimal paths are color-coded in red. Arc tuples consist of the selected job and the increase in the makespan, the finishing time of the last job on the last machine.

C, consisting of nodes $v \in V$ corresponding to *partial solutions* of C and labeled, weighted arcs $a \in A$ representing the assignment of at least one unassigned decision variable to an element of D. The root node r specifies the empty partial solution, the set of terminal nodes T corresponds to complete solutions, and is by construction isomorphic to S.

If the objective function is separable, an arc carries the costs of the associated assignment as weight and we have meaningful costs-so-far for search nodes $g\colon V \to \mathbb{R}$, where $g(v)$ corresponds to the shortest path length from r to v. Otherwise, the arcs leading to terminal nodes carry the weight corresponding precisely to the solution cost of the terminal node and all other arcs before have zero weight. The set of optimal terminal nodes T^* consists of those v^* for which $g(v^*) \leq g(v')\ \forall v' \in T$ and correspond to the optimal solutions. Furthermore, a function $h\colon 2^{\{1,\dots,n\}} \to \mathbb{R}$ estimates the costs of the subproblem of optimally assigning values to the yet unassigned decision variables.[4] Common for h is a lower bound function or the value of a heuristic solution to the subproblem.

The space of partial solutions which is represented by a search graph can be extended to a state graph, where each partial solution induces a state and multiple partial solutions can map to the same state. The state determines the set of feasible completions to all

[4] $2^{\mathcal{X}}$ denotes the powerset of a set $\mathcal{X}$, the set of all subsets of $\mathcal{X}$

Table 4.1: State-space formulations for the three example problems. Costs are per action, the two different objectives *flowtime* and *makespan* are shown for the PFSP, goal is a condition for a state to be goal state. For the MISP all states are goal states since all independent sets are feasible solutions.

Problem	State	Actions	Costs	Initial	Goal
PFSP	jobs left to schedule $\bar{J}$, earliest starting times $\mathbf{t}^e$	select next job $j \in \bar{J}$	*flowtime*: finishing time of j on machine m *makespan*: machine m's time marker's increase	$\bar{J} = J, \mathbf{t}^e = \mathbf{0}$	$\bar{J} = \emptyset$
TTP	games left to schedule $\overline{\mathcal{G}}$, for each team: positions, streaks, forbidden opponents	first select pivot team then select permitted game for pivot	travel distance for team and its opponent	$\overline{\mathcal{G}} = \mathcal{G}$, all teams at home no streaks/forbidden opponents	$\overline{\mathcal{G}} = \emptyset$ all teams at home
MISP	free vertices $\bar{V}$	first select $v \in \bar{V}$, then either exclude v or include v and exclude its neighbors $N(v)$	0 or 1	$\bar{V} = V$	any state

the partial solutions inducing the state. Isomorphic substructures can be detected by checking states for equality and keeping only non-dominated search nodes.

4.3.1 Permutation Flowshop Problem

As first example problem, we choose the permutation flowshop problem (PFSP) with flowtime minimization. We are given n uninterruptible jobs $J = \{j_1, \ldots, j_n\}$, each of which needs to flow through m machines in a predefined order. Each machine can process one job at a time, has a specific processing time p_{ij} for each job j on machine i, and can only start a job when it has been finished by the previous machine. The goal is to find a permutation of the jobs, so that the total flowtime, the sum of finishing times on the last machine over all jobs according to this schedule, is minimal. An example schedule of an instance with $n = 4$ jobs, $m = 3$ machines, and processing times $p_{ij} = \left(\begin{smallmatrix} 4 & 1 & 2 & 2 \\ 2 & 3 & 1 & 4 \\ 1 & 4 & 1 & 2 \end{smallmatrix}\right)$ is shown in Figure 4.1.

Following our formalization with partial solutions, the finite domain $D \equiv J$ are the jobs and the decision variables $x_l, l \in \{1, \ldots, n\}$ assign a job to a position l, where each job can be selected at most once—only permutations of D are valid schedules. The terminal nodes have all n jobs selected and are therefore isomorphic to S, all permutations of J. The objective function is separable and the costs of a partial solution $\mathbf{d} \oplus d_k = (x_1 = d_1, \ldots, x_{k-1} = d_{k-1}) \oplus d_k$ can be defined recursively:

$$g(\mathbf{d} \oplus d_k) = g(\mathbf{d}) + \tilde{t}_{m,d_k} = \sum_{l=1}^{k} \tilde{t}_{m,d_l} \tag{4.1}$$

$$\tilde{t}_{i,d_k} = \max(\tilde{t}_{i,d_{k-1}}, \tilde{t}_{i-1,d_k}) + p_{id_k} \tag{4.2}$$

$$\tilde{t}_{0,j} = 0, \tilde{t}_{i,0} = 0, \tag{4.3}$$

$$\forall i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\} \tag{4.4}$$

where the binary operator $\oplus$ extends a partial solution $\mathbf{d}$ by assigning the next remaining unassigned variable $x_k = d_k$, $\tilde{t}_{i,j}$ is the finishing time of job j on machine i, including a dummy machine 0 and job 0 with finishing times 0 for notational convenience.

A related state-space formulation for the PFSP and the other example problems is listed in 4.1. A state for the PFSP consists of the jobs left to schedule $\bar{J}$ and the earliest starting times on each machine, the *time markers*, $\mathbf{t}^e$. An example state graph for the PFSP with makespan objective is shown in Figure 4.2.

4.3.2 Traveling Tournament Problem

The traveling tournament problem (TTP) deals with finding a double round robin tournament given n' teams and a symmetric distance matrix δ between the teams' venues. It is assumed that teams start and end at their home venue and travel round by round directly from one game venue to the next. The total traveling distance over all teams has to be minimized. According to the *no-repeat* constraint, teams are not allowed to play back-to-back against each other, and the *at-most* constraint, teams are allowed to play at most $U = 3$ games consecutively at home or away.

The TTP can also be formulated as a mixture of a permutation and a subset selection problem, where the finite domain are all games $\mathcal{G} \ni g = (t, t') \in \{1, \ldots, n'\} \times \{1, \ldots, n'\} | t \neq t'$ and we decide round by round which games to play. The number of games is therefore $n = n'(n' - 1)$. Many permutations of games are not feasible solutions, since they are not a double round robin tournament or violate the two additional schedule constraints. Furthermore, many permutations correspond to the same solution, since the ordering within a round has no meaning—this is the subset selection part.

The objective function is again separable and we can assign costs to a partial solution recursively by adding the actual travel distances for involved teams t, t' to play a selected game $d_k = (t, t')$ in round $r \in \{1, \ldots, 2(n' - 1)\}$:

$$g(\mathbf{d} \oplus (t, t')) = g(\mathbf{d}) + \delta(\text{pos}^{r-1}(t'), t) + \delta(\text{pos}^{r-1}(t), t) \tag{4.5}$$

$$+ \mathbf{1}_{r=2(n'-1)} \delta(t', t), \tag{4.6}$$

where pos^{r-1} maps to the team's position in the previous round $r - 1$ and $\text{pos}^0(t) = t$. Similar to the traveling salesperson problem, after a two teams' final game the away team t' has to return from t to its home venue and this travel distance has to be added as achieved by $\mathbf{1}_{r=2(n'-1)} \delta(t, t')$. A heuristic guidance h can be provided in form of a lower bound by summing the solution lengths of optimal tours under capacity constraints to the remaining away games over all teams independently ([46, 155], also Section 3.6).

The key aspects of the related state-space formulation are summarized in Table 4.1 and have already been presented in great detail in Section 3.4 of the previous chapter.

4.3.3 Maximum Independent Set Problem†

In the maximum independent set problem (MISP), we are given a graph $G' = (V', E')$, with vertices V', $|V'| = n$ and edges E', $|E'| = m$. The goal is to find a subset of vertices $I \subset V'$, where for each pair $v, w \in I \implies (v, w) \notin E'$ holds—an eponymous *independent set*—with maximum cardinality.

It is a subset selection problem, which we model with binary decision variables $x_i, i = 1, \ldots, n$ over all vertices, where $x_i = 1$ denotes the selection of a vertex i, $x_i = 0$ the explicit exclusion. Only those assignments are feasible without a corresponding edge in E' between any vertex pair in I. Every path from the root to an arbitrary node induces

a corresponding state, a set of vertices $\bar{V}$ for which no decision has been made yet and which are not adjacent to any previously selected vertex—the so-called *free* vertices [5].

The objective function amounts to the number of selected vertices and is trivially separable—at each layer it is either increased by one if we select a vertex, or zero otherwise. A detailed discussion of this state space formulation in the context of decision diagrams is presented in the works of Bergman et al. [12, 13, 15].

As mentioned before, the corresponding states are identified by the free vertices $\bar{V}$, i.e., the maximal subset of vertices, for which every vertex could still be added to every independent set (encoded by a path) that has led to that state. We encode our solutions as a sequence of binary decisions. A specific aspect is so-called *zero-suppression*. When a free vertex $v \in \bar{V}$ is selected, all of its neighbors $N(v)$ cannot be included anymore in any of the feasible extensions of the corresponding state. We can therefore implicitly set the corresponding decision variables to zero and have performed multiple decisions per arc, thereby saving us the explicit, already implied decisions later. Note that solutions are not anymore encoded by paths of equal length as for problems before and that in each layer a potentially different set of decision variables is involved depending on the set. *Remark:* Comparing this with decision diagrams, we break with the assumption that every layer corresponds to one decision variable but allow more general decisions—actions in a state-space search setting. The main reason for this is that we apply frontier search and cannot make use of long-arcs that skip layers.

Another potential optimization is to enforce 1-decisions due to optimality considerations. For this we have to keep track of the residual degrees of the free vertices, i.e., the number of free neighbors. If a free vertex has no free neighbor (an isolated vertex), a residual degree of zero, then it does not make sense not to include it. Likewise, if it has one free neighbor (a leaf vertex), we can include it, since we cannot find strictly better feasible extensions by excluding it. In the setting of beam search, this optimality reasoning cannot be directly applied [19] and needs to be empirically checked.

4.4 Parallel Beam Search

In this section, we present the algorithmic details of our framework and also implementation details which we observed are important to achieve higher parallel efficiency in Julia. As mentioned multiple times (Sections 2.2, 3.5, 4.2), the main idea of beam search is to traverse a search graph layer by layer and keep in every layer the β most promising nodes and therefore consider many partial solutions in parallel. If the evaluation function, also called guidance, to rank the node runs in polynomial time, so does the whole construction algorithm. Often the nodes carry additional state information to facilitate an incremental evaluation of its successors. Relevant parts determining the runtime are said evaluation of successor nodes, determining the β best ones, and performing the actual transitions to the next layer.

Our goal is a parallelized layer-by-layer traversal of the search graph, splitting the

workload evenly between cores to reach high parallel efficiency. In the spirit of frontier search [96], we only keep the current and the next layer of the search in memory, but with the partial solution directly encoded in the node. We make use of intra-node data parallelism and aim to reduce sequential code part runtimes to achieve high parallel efficiency. The key steps and ideas of our parallel beam search are the following:

- We keep the nodes of the current layer, the next layer, and information about the successor nodes in fixed-length preallocated shared memory regions.
- Each node carries the costs-so-far, the layer number, a fixed-length encoding of the corresponding partial solution, and optional state information/auxiliary data to facilitate incremental evaluation.
- Since the memory layout is static and updates are performed by copy & in-place modification, a node is active if its layer number equals the current layer of the search, otherwise inactive.
- The at most β search nodes are split evenly at each layer among the threads for the evaluation of their successors, assuming an approximate homogeneity over the resulting workload.
- As a result, threads write the incremental information about the successors into their preassigned memory region, most importantly the *priority* $g + h$ (the costs-so-far plus the costs-to-go estimate) to determine a nodes' rank, and the decision leading to the successor (like the selected next job for the PFSP).
- A histogram is created by data parallelism over the successor priorities and sequentially cumulated to identify the β best successors. If there are ties in the bin where the successor with rank β lies, they are sorted by a sequential partial quicksort. See Figure 4.3 for an illustrative example.
- Again with data parallelism, the next layer is created by copying over all nodes with surviving successors and performing the actual transitions as defined by the successor information.
- Every layer is checked for terminal nodes to keep track of the currently best objective value and solution.

The fixed-length representation of solutions and static memory layout allows a great reduction of heap operations and therefore the work of the garbage collector in Julia.[5] Each incremental successor information is stored in a predefined slot of its preallocated memory region, depending on the index of its parent in the beam. The storage is potentially sparse since fewer than the maximum number of successors might be stored, depending on the parent state. For instance, in the TTP a successor might be discarded

[5] https://docs.julialang.org/en/v1/manual/performance-tips/

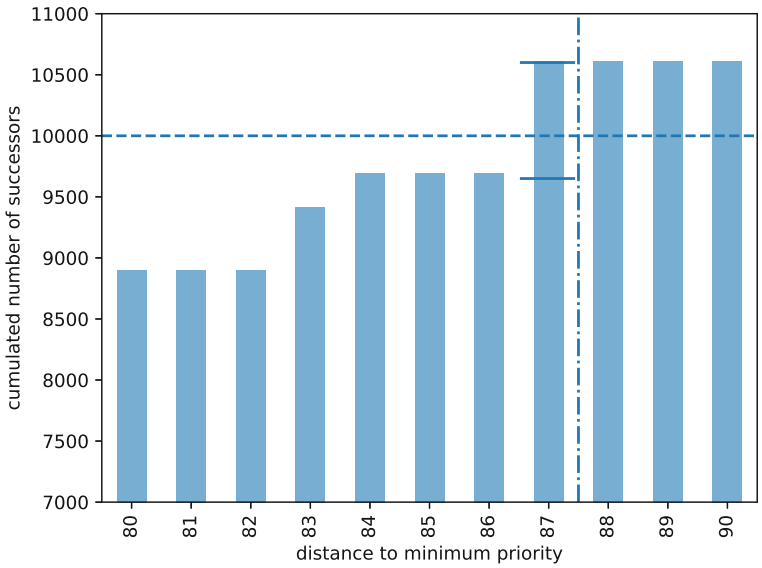


Figure 4.3: Cumulated numbers of successors binned by rounded down priorities with beam width $\beta = 10\,000$ for an exemplary layer. At a distance to the minimum priority of 87 more than β successors are present, therefore all successors with priority ≥ 88 will be discarded, those with ≤ 86 kept. At 87 we perform a tie breaking by partial sorting to select the best ones that lead to exactly β successors.

since a check reveals that it has no feasible completions and in both problems, the number of possible successors decreases by one for each layer. Whether a slot in an array is active is checked by comparing its layer number with the current layer. The storage of the nodes in the beam is dense due to their comparably larger memory footprint.

Since we want to increase the beam widths in the range of millions with a branching factor between two and three orders of magnitude, also the partial sorting to filter the β best successors could become a sequential bottleneck. This is why we suggest a data parallel histogram approach with integer bins, similar to counting sort. The key assumptions for this to work well are that the range of the priorities is small when compared with the maximum number of successors, it is quasi-constant w.r.t. the beam width β, and the number of ties in the cut-off bin remains small relative to the total number of successors. If this is not naturally the case, then the bin sizes could be dynamically adapted instead of using integer bins, which we leave as future work.

For some problems, making use of *dominance filtering* within beam search has shown to be beneficial. There, dominated partial solutions are not pursued further, i.e., those for which another partial solution in the beam exists whose feasible extensions are a superset and whose costs-so-far are not worse. See for example Blum et al. [19] in the case of the longest common subsequence problem. A drawback is the quadratic growth, if we were to check all pairs of nodes or successors. A restriction is to check only for nodes with duplicate states where hashing can be employed for amortized linear runtime. In this work, we focus on parallelized detection of duplicate states for the TTP.

Algorithm 4.1: Data-Parallel Beam Search Algorithm

Input: instance C, auxiliary data A, beam width β, guidance function h, noise parameter σ, beams $Q_{\text{up}}, Q_{\text{down}}$, successor region Q_{S}, dominance rel. $\preceq$, dual bound function b, global primal cut c

Output: feasible schedule $\mathbf{d}$

```
1  d^best = (), Q ← Q_up, Q_next ← Q_down, n_Q ← 1;
2  initialize root node in Q[1];
3  for l ← 1 to l_max(C) do
4  |  optionally-filter-dominated-nodes(C, A, Q, ⪯);
5  |  n_S ← create-successors(C, A, n_Q, Q, Q_S, β, h, σ, b, c);
6  |  n_Q ← process-successors(C, l, n_S, Q_S, Q, Q_next);
7  |  if n_Q = 0 then
   |  |  // no remaining successors
8  |  |  return d^best;
9  |  end
10 |  d^best ← check-for-terminals(C, Q_next, n_Q, d^best);
11 |  Q, Q_next ←flip-beams(Q_up, Q_down);
12 end
13 return d^best;
```

In Algorithm 4.1, we give a high-level description of the intra-node parallel beam search procedure. It receives the preprocessed instance C and corresponding auxiliary data A as input, along with preallocated memory regions for the nodes and the successors $Q_{\text{up}}, Q_{\text{down}}, Q_{\text{S}}$, respectively, and search parameters beam width β, the guidance function b, and a noise parameter σ. The latter adds zero-mean Gaussian noise with standard deviation σ to gradually randomize the ranking of the successors. Optionally, a dual bound function b and a global primal cut derived beforehand from a feasible solution enables pruning of search nodes, for which the dual bound is not strictly better than the global cut. The algorithm returns the best-found solution or the empty solution if no solution was found, which can happen for constrained problems. The parallelized functions *create-successors* and *process-successors* call functions themselves to evaluate successor nodes and make the actual transitions on concrete data types, which is the problem-specific part that has to be implemented by the user of the framework. Both functions return the number of successor information entries evaluated and the number of actually created successors after the selection. We flip between two preallocated beams, each representing alternatively the current or the next layer.

Optionally, depending on the settings of our beam search for a concrete problem, we filter dominated nodes in the parallelized *optionally-filter-dominated-nodes* function in the beam before creating successors, to make space for successors from non-dominated parents. Note that this could in principle already be performed one step earlier when creating the successors. But since we make use of incremental evaluation for performance

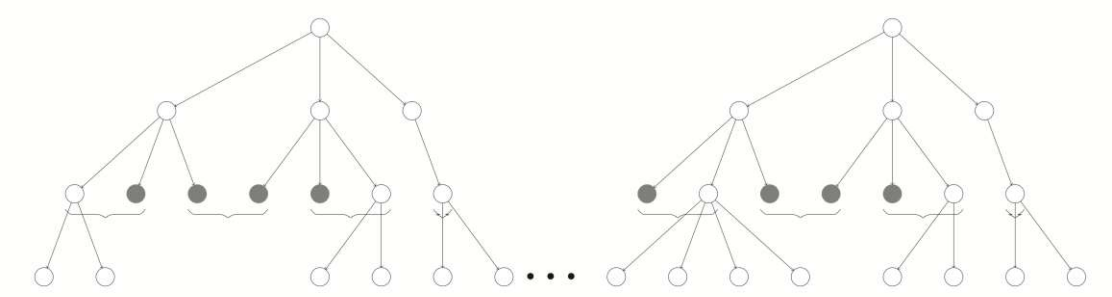


Figure 4.4: Conceptual visualization of the straightforward randomized multi-start approach with multiple distributed workers. The independent runs are parallelized by statically sharing the workload of each layer among the threads.

reasons, we do not have access to the state of the successors yet, since we avoid potentially costly expansions of low-ranked successors.

To distribute the work across threads, we make use of the Julia *Threads* module, which allows loop parallelization with static scheduling of consecutive chunks of data to threads. A data-parallel pattern is employed, where each thread writes into its own private memory region and, if necessary, those are combined in a final sequential step to achieve the desired result, e.g., to calculate the minimum and maximum over an array of numbers. A pitfall is *false sharing*, where threads interfere with each other due to overlapping cache lines of seemingly independent memory regions. Furthermore, Julia stores arrays in column-major ordering, hence for two-dimensional arrays, the thread dimension has to be second and a safety buffer of a cache line length is added between the data.[6] We observed that the histogram binning would otherwise not parallelize at all with even slightly increasing runtimes when multithreaded.

We embed this procedure in a straightforward inter-node parallelization where multiple such randomized runs are statically scheduled on independent MPI workers (visualized in Figure 4.4) over which the best result is then collected and returned. In a start-up phase, the problem instance is preprocessed and potential auxiliary data structures are prepared to facilitate faster/incremental evaluation of successors.

4.5 Computational Study

We implemented the framework in Julia 1.7 and made the source code available on GitHub.[7] In this section, we present computational results on the parallel efficiency and speedup for our implementations of parallel beam search solvers for the PFSP, the TTP, and the MISP. All experiments were run on a cluster with single-socket AMD EPYC 7642 machines each with 48 cores and uniform memory access on 512 GiB of RAM.[8] For

[6] https://juliafolds.github.io/data-parallelism/
[7] https://github.com/nfrohner/parbeam
[8] https://www.grid5000.fr/w/Lyon:Hardware#neowise

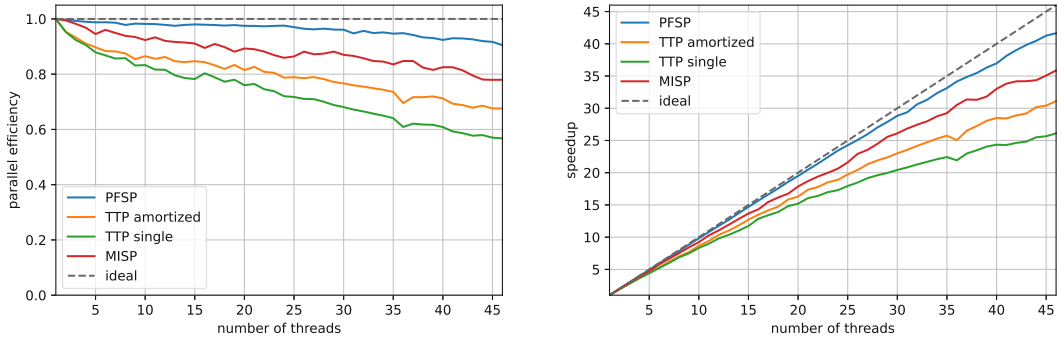


Figure 4.5: Parallel efficiency and speedup over the number of threads for a PFSP parallel beam search on a VFR instance with 100 jobs, 60 machines, and a beam width of 640 000, for a TTP parallel beam search on NL16 instance with 16 teams and beam width 2 000 000 shown without precalculation of the disjoint pattern database, and a MISP parallel beam search on DIMACS instance C2000.9 with beam width 30 000.

medium to large instances, we were able to obtain speedups for single runs in the range of 30-42× using 46 threads (sparing two cores). Furthermore, we obtained many new best feasible solutions for the TTP with large beams on high memory machines within reasonable time by making nested use of intra-node and inter-node parallelization with diversification.

For a fair comparison with ahead-of-time compiled languages, all runtimes are presented without Julia's compilation time by performing a start-up run on a tiny instance before running the actual instance, not included in the measurement. In a production environment, repeated compilation can be avoided by keeping the Julia process running after such a start-up phase.

4.5.1 Permutation Flowshop Problem

We implemented a single-run variant of the iterative widening beam search approach by Libralesso et al. [101] for the flowtime variant of the problem within our framework. With the focus on parallelization and jointly implementing the basics of the framework, we did not implement such an anytime variant for the PFSP which performs multiple runs on the same instance while geometrically increasing the beam width and making use of pruning by the best global upper bound known so far. For the subsequent problems, the TTP and the MISP, we also study such an iterative variant, extending our framework.

More concretely, in each layer, all possible extensions (additions of one from the remaining jobs) are incrementally evaluated and the β best according to the guidance are expanded in the subsequent layer, which is repeated until complete solutions are reached. As guidance h to rank the extensions, the costs-so-far g, i.e., the sum of the finishing times of the so-far scheduled jobs on the last machine, and the total idle time over all machines are combined by means of a weighted sum. The weight for the former is the fraction of jobs scheduled and for the latter the fraction of remaining jobs times the layer number divided by the number of machines.

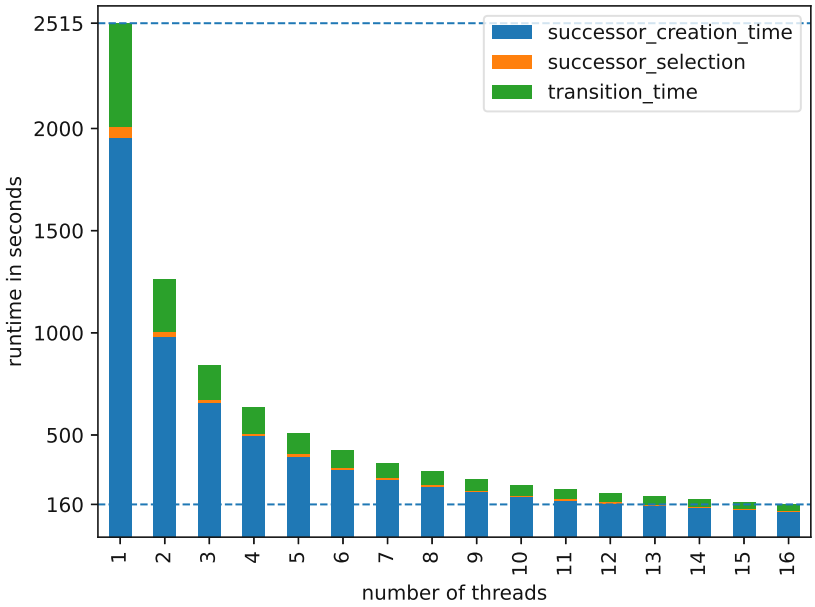


Figure 4.6: Actual distribution of the runtime over the most relevant algorithmic parts, successor creation, filtering, and transition for PFSP parallel beam search on a VFR instance with 100 jobs, 60 machines, and a beam width of 640 000.

In Figure 4.5, we see the parallel efficiency $T_1/\tau T_\tau$ and the speedup T_1/T_τ over the number of threads τ, where T_τ is the runtime of one beam search run with τ threads. From the recent VFR benchmark [156], we consider a medium-sized instance with 100 jobs, 60 machines, and a beam width of 640 000, where we measure a single-threaded runtime of 2516 seconds and a runtime with 46 threads of 60 seconds, resulting in a parallel efficiency of 90% and a speedup of over 41×.

Some parts of the concrete implementation of the algorithm admit a better parallel efficiency than others. Depending on their relative contribution to the overall runtime, which itself depends on the algorithmic parameters like the guidance function and the beam width, these parts determine the shape of the parallel efficiency. For instance, if a parallelized part is memory-I/O heavy or the workload is not well distributed over the threads to keep them busy, then we observe more loss of parallel efficiency. The potentially runtime dominating parts of our algorithm are the instance preprocessing, the successor evaluation, the selection of the β best, the filtering of dominated nodes, and performing the actual transitions. In Figure 4.6 we see the runtime distribution over the first 16 (for better visibility) threads for the example instance from before. The largest portion of work, the successor creation, admits a high parallel efficiency of 94%, followed by the transition parallel efficiency of 89%. The selection drops down to 42%, but itself is a only small part of the runtime resulting in the total parallel efficiency of 90%. Instance preprocessing is negligible (below half a second) and dominance filtering is not performed.

Table 4.2: speedups with 46 threads each pinned on its own physical core measured with single runs for VFR instances with different beam widths β, number of jobs n and machines m.

n	β / m	10k	20k	40k	80k	160k	320k
50	10	10.5	16.1	15.5	15.7	21.3	28.3
	20	12.7	18.8	18.5	19.4	26.9	34.8
100	20	16.2	22.0	22.5	27.9	31.9	37.0
	40	18.1	23.3	27.2	30.9	34.2	40.3
	60	20.5	26.5	30.7	33.8	35.7	41.5
200	20	21.9	25.7	26.4	31.1	33.3	39.0
	40	26.8	28.7	31.9	34.8	35.0	41.6
	60	28.2	30.4	35.3	36.5	36.3	42.0
400	20	27.3	27.9	30.1	31.3	34.0	39.7
	40	32.3	32.9	33.9	35.3	35.9	41.8
	60	33.1	34.4	36.4	37.1	37.0	41.7

Table 4.3: Solution qualities u with a beam width of 320 000, and rounded runtimes t in seconds using 46 threads for the first VFR instance of each group.

n	50		100			200			400		
m	10	20	20	40	60	20	40	60	20	40	60
u	84 212	121 401	367 712	517 049	645 544	1 207 649	1 576 878	1 899 743	4 283 142	5 165 345	5 973 412
t	2	3	11	20	30	42	75	111	160	286	428

To better understand the impact of the instance and algorithmic parameters (the number of jobs n, of machines m, and the beam width β) on the parallelization, we performed runs with different such configurations and measured the speedup with 46 threads. We use the first VFR instance of each group of 11 different combinations of the number of jobs and machines. The results are shown in Table 4.2. For smaller instances and configurations the speedup is limited by Amdahl's law [3], but still already in the double digits. We observe that increasing m has a positive impact on the speedup since it increases the workload of the successor evaluation alone. Increasing the number of jobs n and beam width β increases the overall workload for all parts, which is beneficial since selection is not yet a bottleneck. Lastly, we show in Table 4.3 the concrete solution qualities, the total flowtimes, for the considered instances together with runtimes using a beam width of 320 000 and 46 threads.

4.5.2 Traveling Tournament Problem

We base the implementation of our parallel variant on the beam search for the TTP from Frohner et al. [57]. The latter makes use of a max-heap for the successors, incrementally evaluates successors, and creates them (performs the transition) only when they are better than the worst so far successor if the max-heap has already β successors. A potential advantage is that the memory footprint can be kept lower. In a first preliminary test, we tried to parallelize this by locking the shared resource, the beam, but this was not

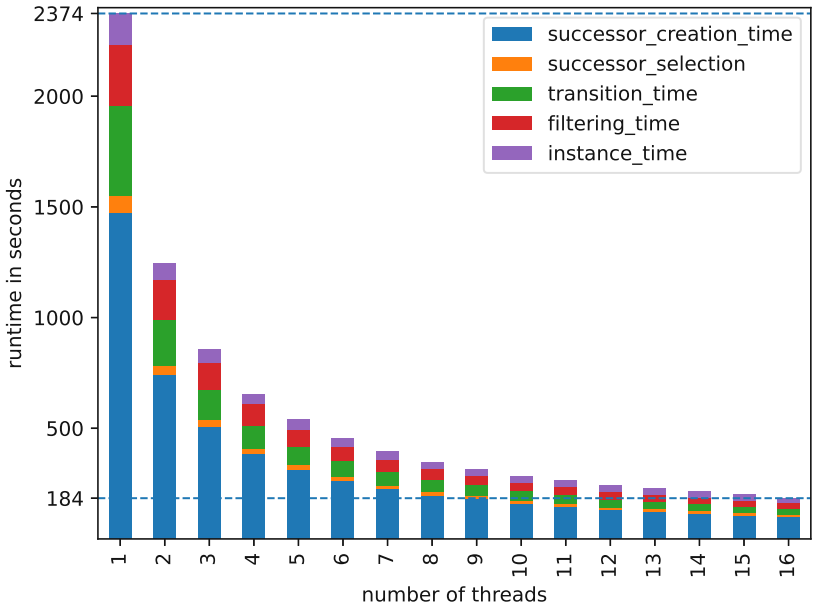


Figure 4.7: Runtime distribution up to 16 threads for TTP parallel beam search on NL16 instance with 16 teams and beam width 2 000 000.

promising, since the critical region appeared to be too long. Therefore, we moved to a data-parallel variant, where the information on the incremental evaluation information (the layer, priority for determining the rank, its parent, and the move leading to the successor) for all successors is saved sparsely in a large preallocated array of size β times the maximum branching factor of the instance.

The schedule is constructed round by round and game by game employing symmetry breaking selecting a pivot team (the one with the smallest number which has not played in the current round) since the order in which games are scheduled in a round is not important. The beam search is guided by the sum of the costs-so-far g, i.e., the travel distances of the teams so far, and the independent lower bound for states as introduced by Uthus et al. [155] for exact tree search approaches applied to the TTP. To calculate this bound, for each team the subproblem of visiting its remaining opponents with minimal travel distance given its current position and streak length while respecting the *at-most* constraints and the number of home stands required for feasibility, but ignoring all other teams' schedules, is solved and the corresponding solution lengths are summed up. Furthermore, for each node's possible games to be played it is checked whether it would lead to another team being without a feasible game to play in this round, and therefore allows detection of dead-ends in the construction earlier. Also, it is checked whether enough home and away games would be left to, in principle, accommodate the remaining away and home games due to maximum streak lengths.

We implement duplicate state filtering to detect nodes with isomorphic feasible extensions by a fixed size hash table with lists of non-dominated states using as the size the closest prime smaller than β. The position of a node in the table is determined by using Julia's

Table 4.4: Impact on mean solution quality $\bar{u}$ (non = no state filtering, dup = duplicate state filtering) and mean runtimes $\bar{t}$ with 32 threads on filtering duplicate states for random TTP instances with team sizes $n' \in \{8, 10, 12\}$, 30 instances each, using a beam width of 100 000. The rate of filtered nodes as fraction of expanded nodes is r_f and $\#u^\mathrm{dup}_{<,>}$ counts the number of strictly better (or strictly worse) solutions over the instance groups using duplicate state filtering.

n	$\overline{u^\mathrm{non}}$	$\overline{t^\mathrm{non}}$	$\overline{u^\mathrm{dup}}$	$\overline{t^\mathrm{dup}}$	r_f [%]	$\#u^\mathrm{dup}_<$	$\#u^\mathrm{dup}_>$
8	32058.8	1.2	32037.0	1.4	3.5	7	0
10	49405.5	1.7	49274.0	2.0	1.9	14	0
12	71488.5	3.0	71236.4	3.4	1.2	19	0

`hash` function on the state data modulo this hash size. Each list is associated with a spinlock to manage thread-safe concurrent access. Since the number of duplicate states for the TTP is small and the critical region is short, this is assumed to parallelize well. On the conditional insertion of a new state into the list of a slot, we iterate over the existing states and check each for dominance. If an existing state is dominated (equal state and strictly worse priority), then it is replaced in the list and deactivated—if it dominates the new state, the latter is deactivated and not added to the list. If neither occurs it is appended to the list of currently non-dominated states. In Table 4.4 evidence for the positive impact on the solution quality of duplicate state filtering is provided. Parallel beam search runs on artificial Euclidean test instances from Frohner et al. [57] provide at least as good or better solution qualities with an acceptable increase in runtime. For team sizes $n' = 8, 10, 12$, solutions are never worse and 7, 14, and 19 times out of 30 each are strictly better ones provided at the cost of 10–20% runtime increase. Therefore, we activate it for all further experiments.

To calculate the lower bounds quickly during construction an exponentially growing lookup table per instance is calculated and stored compressed in a file upfront by constructing a directed acyclic graph separately for each team representing all possible single-team states for the given instance and solving restricted shortest path problems. With one core per team and multiprocessing, this is possible on our cluster for 16 teams instances within 20 seconds, for 18 teams within 120 seconds—for 20 teams we use a slightly relaxed variant of the bound ignoring the number of remaining home games which is then calculable within less than 7 minutes.

For the medium to large sized instance NL16[9] with 16 teams, we observe a behavior not as strong as for PFSP test instance, as the parallel efficiencies of all parts decline, resulting in total parallel efficiency of about 57% for 46 threads, a speedup of 26×, or an amortized parallel efficiency of 67% with corresponding amortized speedup of 31× excluding the bounds precalculation time, see Figure 4.5. The latter consideration is meaningful since for the beam search to be competitive we later employ a multi-start procedure over distributed independent workers randomizing the beam search by adding noise to the guidance. There, the precalculation needs to be performed only once for all runs.

[9] Instances/solutions repo at `https://www.sportscheduling.ugent.be/RobinX/index.php`

Table 4.5: Solution qualities of parallel beam search runs as relative percentage deviation from the currently best-known results u^{ρ}_{rel} and runtimes in minutes $t^{\rho,w}_{\tau}$, where ρ is the number of randomized runs and w the number of independent MPI workers. New best feasible solutions are printed in bold.

inst	u^{1}_{rel} [%]	$t^{1,1}_{46}$ [']	u^{32}_{rel} [%]	$t^{32,8}_{46}$ [']
NL10	0.15	2.3	0.15	9.6
NL12	0.56	4.3	0.42	18.0
NL14	3.60	7.2	1.65	29.8
NL16	0.61	12.2	0.15	49.9
CIRC10	1.65	3.8	0.83	30.2
CIRC12	2.48	6.8	0.50	61.1
CIRC14	1.27	10.7	**-0.96**	110.6
CIRC16	**-0.44**	16.9	**-1.32**	148.8
CIRC18	1.87	25.9	**-1.25**	220.9
CIRC20	0.92	37.8	**-0.35**	284.7
NFL16	1.40	12.0	0.87	49.6
NFL18	2.66	19.8	0.60	77.0
NFL20	3.92	30.8	2.26	105.2
GALAXY10	0.37	2.5	0.29	11.5
GALAXY12	0.84	4.5	**-0.15**	22.5
GALAXY14	0.35	7.6	**-0.01**	37.7
GALAXY16	**-0.07**	12.5	**-0.04**	61.6
GALAXY18	**-0.38**	20.4	**-1.22**	96.9
GALAXY20	0.12	31.5	**-1.43**	129.4
SUPER10	0.01	2.3	0.00	9.5
SUPER12	**-0.15**	4.3	**-0.41**	17.9
SUPER14	**-0.12**	7.2	**-0.48**	30.6
mean	0.98	12.9	0.01	73.3

The reason for the reduced parallel efficiency is likely due to higher variance in the distribution of the load to the workers due to the problem's constraints and generally the less cache-friendly memory-IO since the evaluation of the successors includes random access memory lookups of the precalculated data. The runtimes of the different algorithmic parts up to 16 threads are shown in Fig. 4.7, with a speedup of 13×. This time, also instance preprocessing (precalculation, storing, and loading of bounds) and the duplicate state filtering are relevant and also parallelize sufficiently well.

In a ballpark comparison of absolute runtimes, it is worth noting that the heap-based single-threaded beam search implementation in Julia took about half a day for a run with a beam width of 250 000, while it took our parallel variant only about 12 minutes with 46 cores and an even larger beam width of 20M. This amounts to a gain (speedup × beam width ratio) of three to four orders of magnitude, attributed to a faster single-threaded implementation reducing garbage collector usage and good parallel efficiency.

Having machines available with much memory to go for large beam widths, we make a comparison with the best-known results for a diverse set of well-known instances from

Table 4.6: Exemplary tie rates for beam search runs with beam width 15 M, either with ($\sigma_{\text{rel}} = 0.1\%$) or without noise ($\sigma_{\text{rel}} = 0.0\%$), and the corresponding sequential partial quicksort times $t^{\text{qs}}_{\text{rel},46}$ as fraction of the total runtime with 46 threads. The relative standard deviation σ_{rel} is the fraction taken from the root lower bound to act as an instance's absolute standard deviation of the Gaussian noise.

inst	NL14	NL14	CIRC14	CIRC14	GAL14	GAL14	SUP14	SUP14
ties [%]	2.1	0.2	56.3	34.0	8.5	2.5	1.3	0.1
$t^{\text{qs}}_{\text{rel},46}$ [%]	1.2	0.4	27.6	24.2	3.0	2.5	0.5	0.1
σ_{rel} [%]	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1

the literature, the NL, CIRC, NFL, GALAXY, SUPER benchmark instances, between 10 and 20 teams. We first perform single runs with a beam width of 20 M and 46 threads and provide the relative percentage deviation u^1_{rel} (negative means that we improve the best-known solutions) to the best-known upper bounds and the overall runtime $t^{1,1}_{30}$ in minutes in Table 4.5. We observe in the mean a relative deviation of about 1% and obtain four new best feasible solutions. To go further, we perform 32 randomized parallel beam search runs distributed on 8 independent MPI workers, resulting in approximately 4× the runtime of the single runs. In the right two columns of Table 4.5 we see that this diversification pays off since we find 11 new best feasible solutions out of 22 instances and obtain a mean percentage deviation of close to zero with an average runtime of below 75 minutes per instance. We randomize by adding Gaussian noise to the guidance with zero mean and an instance-dependent standard deviation of $\sigma_{\text{rel}} = 0.1\%$ of the root lower bound. Its effect is a tie breaking and gradual shuffling of the search nodes.

To better understand the runtime dependency on the instances, we study the impact of the tie rate—the number of successors in a cut-off bin divided by the total number of successors—on the potential speedup, since for the tie breaking we make use of a sequential partial quicksort. It is a very efficient sorting algorithm in practice to select the best k elements of a list, still when going to large beam widths and branching factors to sort hundreds of millions of tied successors can become a sequential bottleneck. This is why we use our parallelized histogram approach in the first place. In Table 4.6 we see the fraction of the runtime in percent of the sequential partial quick sort for different exemplary instances for parallel beam search runs with 46 threads and a beam width of 15 M. The impact is limited by up to 3%, except for the CIRC instances, which have additional symmetries since the teams are placed on a circle with unit distance between each other. Also the galaxy instances have somewhat more ties which we believe is attributed to the instances' rather small and more uniformly distributed distances. Those trends on the runtime for different instances with the same number of teams can also be observed in Table 4.5.

Finally, we perform an iterative widening approach [68, 101], an anytime variant of beam search iteratively increasing the beam width and potentially applying pruning by dual bounds. In our case, we perform randomized batches of data-parallel beam search runs with increasing beam widths while making use of the global primal bound to prune the search tree using a dual bound. We set a time limit of 40 minutes and a beam width limit

Table 4.7: Solution qualities of iterative widening parallel beam search runs on a single machine with 46 threads and a time limit of 40 minutes. Beam width increases by a factor 2 after every batch of two randomized runs with $\sigma_{\text{rel}} = 0.1$. The percentage gaps u_{rel} relative to the best-known feasible solution, the best-found solution length u, the log-beam widths of the last run $\tilde{\beta} = \log_2 \beta$, and the times of the last improvement t^{last} in minutes are shown.

inst	best	u	$u_{\text{rel}}[\%]$	$\tilde{\beta}$	t^{last} [']
NL12	110729	111174	0.40	26	17.2
NL14	188728	191927	1.70	25	27.1
NL16	261687	264170	0.95	24	10.9
CIRC12	404	404	0.00	25	18.5
CIRC14	628	628	0.00	25	0.6
CIRC16	910	898	**-1.32**	24	13.8
CIRC18	1284	1284	0.00	23	5.9
NFL16	231483	233805	1.00	24	17.8
NFL18	282258	286375	1.46	24	9.1
GALAXY12	7180	7163	**-0.24**	26	8.7
GALAXY14	10879	10903	0.22	25	28.2
GALAXY16	14648	14598	**-0.34**	24	14.5
GALAXY18	20489	20392	**-0.47**	23	39.1
SUPER12	460870	459115	**-0.38**	26	20.7
SUPER14	571632	568093	**-0.62**	25	23.4

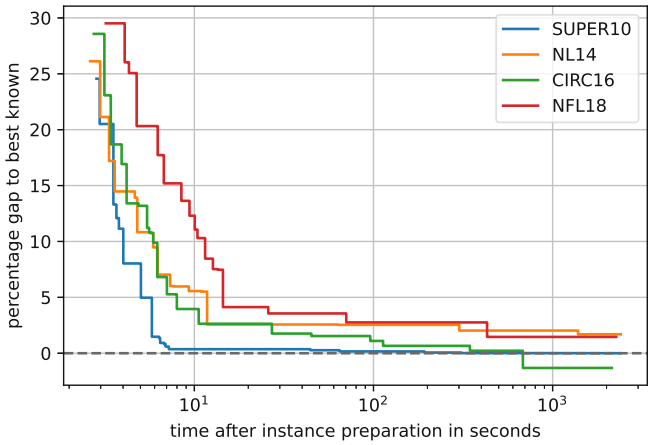


Figure 4.8: Anytime behavior of iterative widening 46-threads parallel beam search runs on selected TTP instances. Solutions with a gap to best-known solutions below 3% can be created rather quickly between 10–100 seconds.

of 2^{27} (whichever is exceeded first), start with a beam width of one and increase it by a factor of two every two runs, based on preliminary manual tuning. As randomization, we perform random variable ordering, random tie breaking in the cut-off bins of the histogram to select the β best successors and add a noise of $\sigma_{\text{rel}} = 0.1$ as before. The results are summarized in 4.7 and show that competitive solution qualities can also be obtained quickly in a single-machine many-core setup that way when time should be the natural termination criterion. Furthermore, the anytime behavior of selected instances is displayed in Figure 4.8. We see that good solutions are found quite quickly while closing

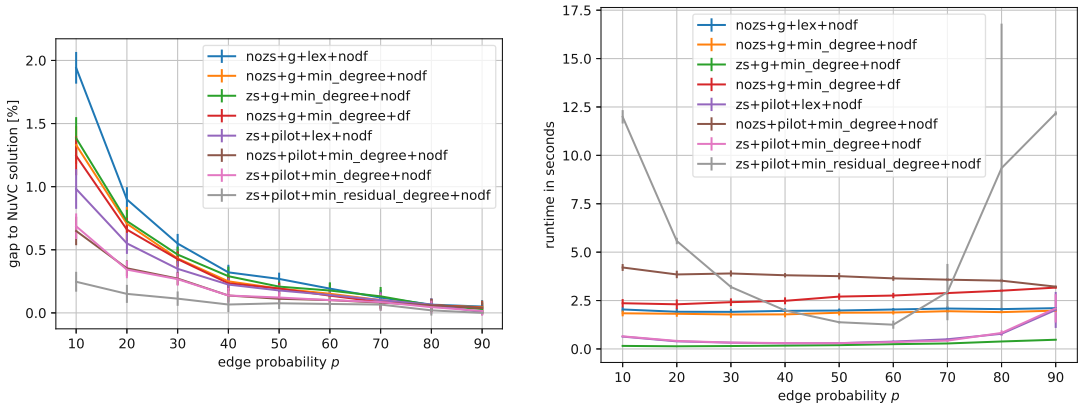


Figure 4.9: Gaps and runtimes to best-found solutions by NuMVC 10-minute runs for different beam search configurations on random graphs over increasing edge probabilities with $\beta = 1\,000$, single-threaded.

the remaining gap remains time intensive and distributing the runs on multiple workers as done before still makes sense.

4.5.3 Maximum Independent Set Problem†

We first study the impact of different combinations of guidance function, variable ordering, zero-suppression, and duplicate detection on solution quality and runtime. To this end, we create a set of random graphs following the Erdős–Rényi $G(n, p)$ model [49], where p is the independent probability for the existence of a given edge. We set $n = 1\,000$ and vary p from 10% to 90% in steps of 10% with 20 instances each. As a baseline, we run each instance with the state-of-the-art local search based NuMVC solver by Cai et al. [27] for 10 minutes and calculate gaps to this baseline for the beam search runs.

The results are shown in Figure 4.9 for $\beta = 1\,000$, single-threaded and 4.10 for $\beta = 10\,000$ using 14 threads. The more difficult graphs are the sparse ones, whereas for the dense the gaps come close to 0%. As guidance, we compare the g-value (the costs so far) with a PILOT-style approach [43], where we evaluate each search node by a greedy rollout using the minimum residual degree heuristics. For variable orderings, we select the next vertex either lexicographically, with a static minimum degree, or dynamically with a current minimum residual degree. We use either zero-suppression (*zs*) or not (*nozs*) and duplicate filtering by hashing (*df*) or not (*nodf*).

Unsurprisingly, we observe that the guidance has the strongest impact. The rollouts appear to provide a good estimate for the optimal feasible completions. In particular, when the rollouts using the minimum residual degree heuristics are paired with the same variable ordering in the high-level search, the gaps are overall below 0.25% for $\beta = 1\,000$ and below 0.2% when increasing the beam width to 10 000. The main benefit of zero-suppression is the large decrease in runtime since fewer node expansions have to be performed, at no or very small loss in quality. The impact of duplicates filtering is quite small and therefore not used in further experiments.

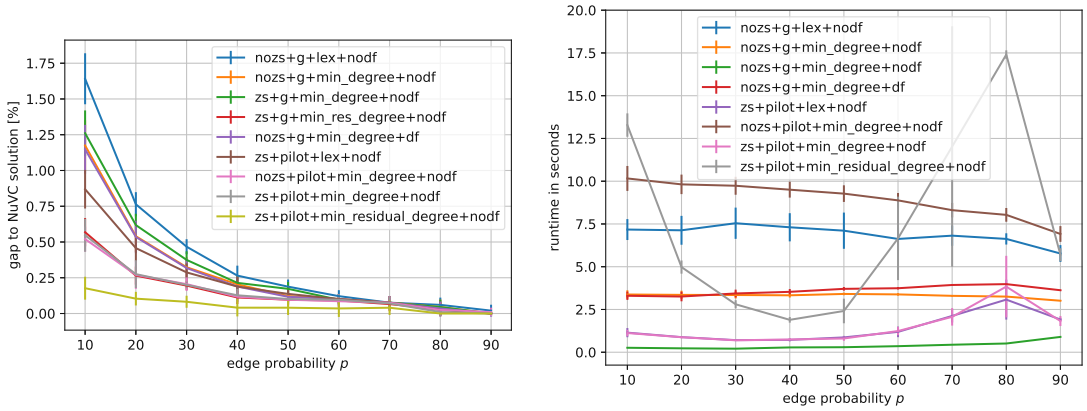


Figure 4.10: Gaps and runtimes to best-found solutions by NuMVC 10-minute runs for different beam search configurations on random graphs over increasing edge probabilities with $\beta = 10\,000$, 14 threads.

As expected the PILOT guidance approach has a substantial impact on runtimes, since for every node the corresponding subproblem is solved by the minimum residual degree heuristic. In the binary branching, it either excludes or includes a vertex and iteratively selects a free vertex with a minimum residual degree to evaluate the remaining subproblem and calculate the f-value. The runtime of such a rollout depends on the data structure used to keep the residual degrees and to select a minimum-degree vertex. The denser the graph, the greater the number of edges and the smaller the independent set. There we observe that keeping an array of residual degrees and selecting the minimum degree vertex by a linear scan is sufficient and the overhead of using a min-heap is beginning from some density not justified anymore. For sparse graphs, the number of edges becomes smaller and the independent sets longer. Using a linear scan then results in a $\mathcal{O}(n^2)$ runtime behavior for the rollout and a heap is therefore preferable with only $\mathcal{O}(n \log n)$.

In Figure 4.11, we compare the (more complicated) runtime behavior of complete beam search runs using the two different data structures in the critical region for random graphs with $n = 1\,000$. At first, the runtime increases when going to smaller edge probabilities, since the search trees become larger as the independent set cardinalities increase. Near the critical point $\log(n)/n$, the graph becomes decomposed into increasingly smaller connected components, which combats this increase. This prunes the search since we always add currently isolated and leaf vertices to the independent set to reduce the branching factor and to some extent emulate preprocessing.

After the algorithmic parameter tuning, we now move to the famous DIMACS benchmark instances[10] [146], where we use the complement graphs as common for the MISP. We first measure the speedup and parallel efficiency on a rather large DIMACS instance, as shown back in Figure 4.5, together with the other problems. It exhibits a parallel efficiency of almost 80% when going to 46 threads, corresponding to a speedup up to 36×. In contrast

[10] https://lcs.ios.ac.cn/~caisw/graphs.html

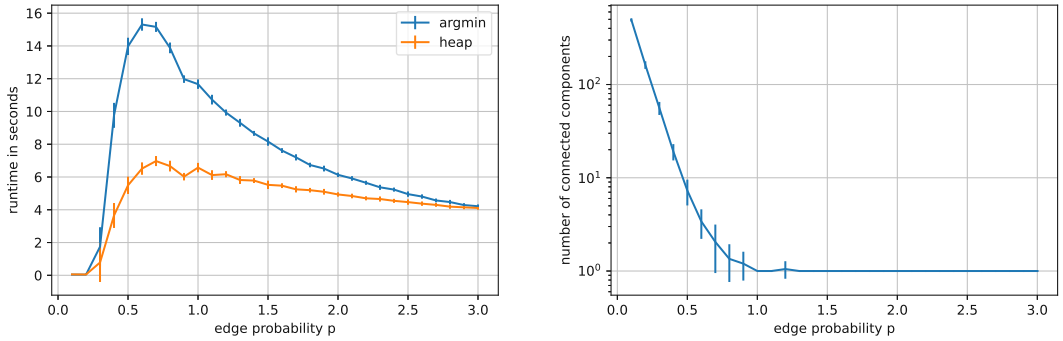


Figure 4.11: Left: runtimes on sparse random graphs with 1 000 nodes and a beam width of 100 over different increasing probabilities with 20 graphs each. We compare the runtime once with a binary heap to select the next vertex with minimum residual degree in the heuristic evaluation in constant time $\mathcal{O}(1)$ but an update factor on every edge of $\mathcal{O}(\log n)$ vs. a simple $\arg\min$ in $\mathcal{O}(n)$ over a residual degrees array with an update factor in $\mathcal{O}(1)$. Right: mean and standard error of the number of connected components, critical point at $\log n/n \approx 0.7\%$.

to the implementations for the PFSP and TTP, almost all the computational effort goes into the successor evaluation, where we perform the greedy rollouts as described before.

In a comparison with state-of-the-art solvers, we employ an iterative widening approach [68, 101], an anytime variant of beam search, where we perform runs with increasing beam widths while making use of the so-far best known global primal bounds to prune the search tree using a dual bound. We set a time limit of 10 minutes and beam width limit of 2^{22} (whichever is exceeded first), start with a beam width of one and increase it by a factor of four every two runs, based on preliminary manual tuning. As randomization, we perform random tie breaking in the cut-off bins of the histogram to select the β best successors. We compare the results with fast iterative local search approaches by Andrade et al. [5] (also based upon Grosso et al. [76], called ILS and GLP), using directly their reported results, and 10-minute runs of NuMVC [27] on our computational test bed. As reported by the authors, ILS and GLP are run 15 times each on a 3.16 GHz Intel Core 2 Duo CPU with 4 GB of RAM, where as termination criterion a maximum number of arc scans is used instead of a time limit.

The results for selected difficult DIMACS instances are shown in Table 4.8. When the best objective value has been found is unfortunately not reported, which makes the comparison a bit more difficult, while as additional information the frequency of the best solutions found over 15 runs for each instance and algorithm is shown in parentheses. We observe that the iterative widening beam search is close but a bit behind the local search approaches in terms of solution quality. We observe diminishing returns for larger beam widths and we believe there is at some point more room for improving the beam search in terms of diversification, e.g., randomizing the variable ordering and guidance, possibly by performing Monte Carlo rollouts [29] instead of applying fixed heuristics.

Table 4.8: Results on selected difficult DIMACS instances, reprinting and comparing results from iterative local search approaches by [5, 76] (ILS, GLP) with parallel beam search (PBS) and NuMVC [27] run on our test beds with a time limit of 600 seconds. Mean times $\bar{t}$ reported for ILS and GLP, last improvement t^{last} for PBS and NuMVC, and last log-beam width employed $\tilde{\beta} = \log_2 \beta$.

				ILS		GLP		PBS-30			NuMVC	
inst	n	density	best	$\lvert I\rvert$	$\bar{t}$	$\lvert I\rvert$	$\bar{t}$	$\lvert I\rvert$	$\tilde{\beta}^{\text{last}}$	t^{last}	$\lvert I\rvert$	t^{last}
C1000.9	1000	0.1	68	68(15)	25	68(15)	50	67	22	40.3	68	0.8
C2000.5	2000	0.5	16	16(15)	436	16(15)	967	16	20	3.9	16	0.5
C2000.9	2000	0.1	80	77(14)	103	79(2)	182	78	20	423.6	78	28.1
C4000.5	4000	0.5	18	18(1)	1897	18(15)	3708	17	18	5.8	18	125.6
DSJC1000.5	1000	0.5	15	15(15)	103	15(15)	258	15	20	2.0	15	0.5
MANN_a27	378	0.0	126	126(15)	1	126(15)	2	126	22	1.2	126	0.0
MANN_a45	1035	0.0	345	345(8)	3	344(12)	5	345	18	13.3	345	18.1
MANN_a81	3321	0.0	1100	1100(15)	10	1098(9)	17	1100	14	50.4	1098	3.9
brock200_2	200	0.5	12	12(15)	4	12(15)	13	12	24	1.1	12	0.0
brock200_4	200	0.3	17	17(15)	4	17(15)	9	17	24	1.9	17	0.9
brock400_2	400	0.3	29	25(15)	11	29(10)	20	25	24	1.4	29	188.5
brock400_4	400	0.3	33	33(10)	11	33(15)	16	33	24	40.9	33	8.8
brock800_2	800	0.3	24	21(15)	60	21(15)	111	21	22	1.3	21	0.3
brock800_4	800	0.3	26	26(1)	60	26(2)	112	21	22	2.0	21	0.4
hamming10-4	1024	0.2	40	40(15)	39	40(15)	98	40	22	2.5	40	0.1
keller6	3361	0.2	59	59(15)	519	59(15)	862	56	18	258.0	59	2.7
p_hat1500-1	1500	0.7	12	12(15)	173	12(15)	808	12	20	80.9	12	0.3
p_hat1500-2	1500	0.5	65	65(15)	150	65(15)	252	65	20	2.0	65	0.1
p_hat1500-3	1500	0.2	94	94(15)	88	94(15)	136	94	20	11.4	94	0.0

4.6 Conclusions and Future Work

We presented a parallel beam search framework for combinatorial optimization problems performing a cooperative construction of the search tree layer by layer using multithreaded data parallelism. We published an implementation in the Julia language together with example implementations for three well-known optimization problems, the permutation flowshop problem (PFSP) with flowtime objective, the traveling tournament problem (TTP), and the maximum independent set problem (MISP), based on state-of-art beam search approaches from the literature. By appropriate algorithmic design and following Julia performance optimization best practices, we were able to achieve speedups between 30–42× with 46 threads for medium to large problem instances and sufficient beam width, and a parallel efficiency in the range of 60–90%. Additionally, we implemented a straightforward inter-node parallelization with the message passing interface (MPI) to perform multiple randomized runs on distributed independent workers. For the TTP, the large absolute speedup when compared to previous implementations allows attacking the problem with very large beam widths up to 20 million on high memory machines with which we found 11 out of 22 new best feasible solutions for difficult benchmark instances with up to 20 teams. Furthermore, we show that an anytime beam search variant, iterative widening parallel beam search, can also provide strong solutions for the TTP in a single-machine multi-core setup within a short time.

For the considered problems the occurrence of search nodes with the same rank according

to our histogram-based sorting—ties— is an important issue to consider. We employed a random tie breaking by adding small Gaussian noise to these values. Alternatively, a second informed criterion (e.g., costs-so-far) could be added with a small factor. In both cases, a smarter binning strategy would be interesting future work to reduce the number of ties in the cut-off bin. The sequential partial sort could be replaced with a, e.g., parallel radix sort. It would also be interesting to compare the efficiency of the Julia implementation with other languages/frameworks focusing on parallelization like OpenMP or Chapel, and further workload scheduling approaches. On the inter-node level, a natural continuation would be to spread the beam over distributed workers, connecting to existing large-scale parallel BFS approaches, e.g., by Buluç and Madduri [26]. Extending the framework to AI and planning problems would also be interesting future work.

A general disadvantage of large-scale beam search over other methods is the notoriously high memory demand. At some point, we observe diminishing returns for larger beam widths and there is more room for improving the beam search in terms of diversification and guidance. Apart from focusing on achieving a high throughput of the search as we did in this chapter, it is at least equally important to improve diversification and guidance of the search, which can often provide a more powerful lever. A recent direction is to learn heuristics using data-driven machine learning based approaches as proposed by, e.g., Huber and Raidl [84] or Choo et al. [30].

CHAPTER 5

Same-Hour Delivery with Fair Tardiness and Flexible Shifts

In the second part of this thesis, we are moving from static to dynamic and stochastic optimization. More concretely, we are facing a fully dynamic vehicle routing problem with stochastic customers motivated by a real-world application where orders arrive at an online store dynamically over a day to be delivered within a short time—a *same-hour delivery problem* (SHDP).

Full dynamism is given since we do not know any orders in advance, whereas the stochastic aspect comes into play by having estimates for the hourly numbers of orders. The goal is to satisfy the daily demand by constructing closed routes from a single depot to the customers given a set of drivers with a predefined shift plan and the hourly demand estimates as input while first minimizing and balancing due time violations across customers and then labor and travel costs. Labor costs are subject to optimization since the end times of shifts have a certain amount of flexibility since we can either dynamically book overtime or send them home a bit earlier.

In this chapter, we present a novel double-horizon approach based on the shifts and the hourly demand estimation. Within the shorter horizon, we optimize the routes for the orders currently available. In the longer horizon, we extrapolate until the end of the day to determine target shift end times for the drivers, which are then fed back into the point-in-time objective function. Furthermore, we devise a route departure time strategy that balances route quality and risking due time violations. The route optimization is performed by an adaptive large neighborhood search (ALNS).

We consider real-world inspired artificial instances and compare the results for full-day online problem simulations with those for the offline scenario where all orders of a day are known in advance. We observe a sensible trade-off between tardiness and costs as compared to extreme fixed route departure time and shift end strategies.

This work has been published in the proceedings of the PATAT 2020 conference and presented at its postponed and accumulated edition PATAT 2022:

> Nikolaus Frohner and Günther R Raidl. A double-horizon approach to a purely dynamic and stochastic vehicle routing problem with delivery deadlines and shift flexibility. In *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling - PATAT 2021: Volume I*, 2020

5.1 Introduction

Motivated by a real-world application where customers place orders at an online store to be delivered within a few hours, we introduce a specific dynamic vehicle routing problem with stochastic customers (DVRPSC) variant called *Same-Hour Delivery Problem with Fair Tardiness and Flexible Shifts*. Orders arrive dynamically over the day, and each order is due only hours after arrival, where the specific due times vary and depend on the orders' types. These orders are picked at a single depot and are subsequently available for delivery to the customers by drivers with predefined shifts.

The goal is to assign the orders to the drivers and perform the routing in a way to avoid or minimize due time violations. Drivers perform multiple routes over the day and for each route a decision has to be made about when to start it. This is crucial since after the departure of a driver, the corresponding route cannot be changed anymore. As a secondary objective, the labor costs, which are determined by the actual shift end times, and the travel times, determined by the performed delivery routes, are to be minimized. The shift end times are subject to some flexibility and may be ended earlier or extended to account for the uncertainty of the actual load.

In particular, we need to account for the strong dynamism of the problem by making use of the stochastic information known in advance. As such, an estimation of the demand for each hour over the day is available upfront. To link this information to the shifts, the time-dependent average number of orders drivers can handle per hour—the driver performance—needs to be estimated. In this work, we combine the well-known ALNS for vehicle routing [117, 127] with a double horizon approach [109] to handle dynamism and stochasticity. In the short horizon planning, we present a driver performance-dependent route departure time strategy—more efficient routes are started earlier than inefficient routes, where improvement is still expected. To avoid sending drivers home too early, we look ahead until the end of the day—the large horizon—by solving a simplified assignment problem on the expected orders without concrete routing to predict target shift end times for the drivers.

In Section 5.2 we discuss related work. The formalization of the different problem variants (offline, point in time, online), the solution representation, and the objective function is presented in Section 5.3. Short-horizon route construction is done by ALNS using classical insertion and regret heuristics and a diverse set of destroy operators as briefly discussed

in Section 5.4. We present the details of our driver performance estimation in Section 5.5 which is crucial for our departure time strategy (Sect. 5.6) and our double horizon approach (Sect. 5.7). The latter is also used to enable informed shift ending strategies as described in Section 5.8. In the computational study (Sect. 5.9), we compare the double-horizon approach with fixed, less sophisticated strategies, on artificial instances with different load patterns (business day vs. weekend) and shift plans (generous vs. tight vs. shortage). We observe substantial advantages of the former. We conclude in Section 5.10.

5.2 Related Work

For general overviews on methods to solve dynamic and stochastic VRPs, see the surveys by Ritzinger and Puchinger [125], Ritzinger et al. [126], Pillac et al. [116], and Psaraftis et al. [121]. Our problem falls into the general class of dynamic VRPs (DVRPs) with stochastic customers [126], i.e., where information is revealed over time with connected stochastic available upfront. Mor and Speranza [111] provide another recent classification of what they call *VRPs over time*, in which the orders need not only to be assigned to vehicles and sequenced but also the starting time of routes is emphasized as an important decision. In the offline view assuming perfect knowledge for a considered time horizon, our problem can be seen as a multitrip VRP with release and due dates (MTVRPRD) [28, 137]. In the online view, focusing on the day as the delivery horizon and timely delivery of highly dynamic customer requests as the primary goal, the problem is best categorized as same-day delivery problem (SDDP, [150, 162]) with soft deadlines (SD, Ulmer [147]). A recent survey from a broader operations research perspective on established and emerging (e.g., autonomous delivery, crowdshipping) last-mile delivery concepts is provided by Boysen et al. [21], discussing also strategic decision making and sustainability aspects while the focus remains on routing and scheduling.

To handle uncertainties existing approaches typically fall into one of two categories: those based on *sampling* and those based on *stochastic modeling* [116]. As their name suggests, sampling strategies incorporate stochastic knowledge by generating scenarios based on realizations drawn from suitable random variable distributions. Each scenario is optimized by solving the implied static and deterministic (i.e., offline) problem variant. Then, a consensus solution is typically derived from all scenario solutions, which is applied in the next time step, until a re-optimization takes place. The advantage of sampling is its relative simplicity and flexibility on distributional assumptions, while its drawback is the massive generation and required solving of scenarios to accurately reflect reality.

On the other hand, approaches based on stochastic modeling integrate stochastic knowledge analytically. They try to formally capture the stochastic nature of the problem and are usually highly technical in their formulations and require to efficiently compute possibly complex expected values. Typically, only strong abstractions from the real world allow for stochastic modeling. Applied methods to solve such stochastic models include Markov models and stochastic dynamic programming. For example, Klapp et al. [92]

consider a dynamic dispatch wave problem (DDWP) assuming a set of potential orders with known locations to be served by a single vehicle from a depot until the end of the day. An a priori policy is derived dictating whether a vehicle either waits or is dispatched with certain orders at discrete decision epochs (waves) by minimizing expected costs in form of travel times and probability-weighted penalties for unserved orders using an integer programming model. In the case of our problem, precise and flexible enough analytical models, unfortunately, appear to be out of reach.

Bent and Van Hentenryck [11] describe an event-based model to solve a dynamic VRP. In their *multi plan approach* (MPA) a set of possible routing plans is maintained at any time and updated at certain events, inspired by previous work from Gendreau et al. [65]. There is one distinguished "best" plan which is determined by an appropriate selection function. The events are new customer requests, vehicle departures according to a current distinguished plan, the availability of newly generated plans, and the timeout of plans. The authors further extend the MPA by sampling to a *multiple scenario approach* (MSA) in order to obtain more robust solutions concerning the stochastic aspects. A number of scenarios are created by adding randomly sampled artificial orders, these scenarios are solved, and then a consensus solution is derived for the original online problem at a certain time. A tabu search is used for actually solving the occurring subproblems. We essentially also follow the fundamental concept of the event-based model of the MPA, although with just one current solution.

Hvattum et al. [85] propose another sampling scenario-based approach in conjunction with a rather simple hedging heuristic. Gendreau et al. [65] describe a parallel tabu search approach with adaptive memory for a dynamic vehicle routing problem. Essentially, an MPA-like event model is used in conjunction with tabu search and the problem is resolved whenever new information is available while solutions are continuously improved in the meantime. Stochastic aspects are not considered here, but the focus lies on an effective parallelization.

Ropke and Pisinger [127] and Pisinger and Ropke [117] proposed *Adaptive Large Neighborhood Search* (ALNS) for more general vehicle routing problems, which is nowadays widely used as a framework for a large variety of optimization problems. ALNS is appreciated for its practical efficiency as well as robustness on many occasions. The main idea of ALNS is to repetitively destroy a current candidate solution partially and repair it in a sensible way. Both are done by using sets of different basic operators, which are typically randomized. Improved solutions are always accepted as new current ones, while worse solutions are only accepted according to a Metropolis criterion. The application probability of the individual destroy and repair operators are adapted over the iterations based on their successes in previous iterations. ALNS is today among the most often applied metaheuristics for VRPs in general, and we find it also most useful as core optimization technique for solving our routing problem, see Section 5.4.

Azi et al. [8] consider a VRP with a particular focus on multiple routes per vehicle, as we also have to do. A major difference to our problem is that here the focus is on deciding upon the acceptance of requests. The solution approach is an ALNS that is in several

aspects similar to those from Ropke and Pisinger. Azi et al. [7] extend this work towards the dynamic problem variant. Multiple scenarios combined of real and sampled orders are maintained to evaluate the profitability of new requests.

Schilde et al. [133] describe a variable neighborhood search metaheuristic for a dynamic dial-a-ride problem. The authors also apply sampling to deal with the stochastic aspects. In their variable neighborhood search the shaking moves bear some similarities with the destroy and repair operations of ALNS.

Mitrović-Minić et al. [109] describe a double horizon approach for solving a dynamic pickup and delivery problem. A large horizon is considered for maintaining routes in a state to be able to easily respond to future dynamically appearing requests, while a short horizon is considered for the actual goal to minimize the route lengths based on the so far known requests. While the considered problem is quite different from ours, we adopt the basic idea of considering two planning horizons in our *double horizon approach* in Section 5.7.

As routes frequently have slack induced by their orders' time windows, waiting strategies are identified as an important component to anticipate future requests. There, they are discussed in several works, e.g., [108, 24, 145, 10, 163], including both theoretical results for problems with limited complexity and empirically studied parameterized heuristic strategies. For a recent overview, see Vonolfen and Affenzeller [163]. In our problem, waiting can only be performed at the depot but not at the customer locations, which amounts to scheduling planned routes as a whole. We compare two extreme strategies, earliest and latest, with a thresholding strategy [86, 163] based on the route efficiency comprising two learnable parameters.

A more recent research direction concerns *same-day delivery problems* (SDDPs), with which we share the customer satisfaction under tight delivery deadlines aspect as being the prime goal. Voccia et al. [162] introduce a general same-day delivery problem with the single objective to maximize the number of accepted customers. They solve it with an aforementioned online scenario sampling approach [11] proposing a new consensus algorithm. Ulmer [147] provides a real-world case study of the impact delivery deadlines have on the costs in the context of same-day delivery. All customers have to be served with the objective to minimize the sum of tardiness over the customers. As a dynamic policy a myopic but fast cheapest insertion heuristic is employed, which starts from the current route plan and inserts the new customer cost-optimally with the potential rerouting of depot visits of the affected vehicle. Ulmer and Thomas [148] propose a policy function approximation approach to quickly decide whether to use in a heterogeneous fleet either a high-capacity/low-velocity vehicle or a low-capacity/high-velocity drone to deliver an order, based on the distance to the depot. It reduces the online computation time by training related parameters in an offline simulation and training phase. Similarly, Van Heeswijk et al. [158] learn for driver dispatching problem a linear value function approximation with basis functions through offline simulations with backward passes to be used online for fast decision making.

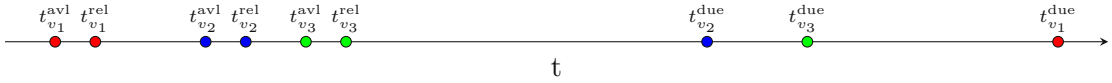


Figure 5.1: Visualization of order-related times of an example route $r = \{0, v_2, v_3, v_1, 0\}$. $t^{\text{avl}}_v \leq t^{\text{rel}}_v \leq t^{\text{due}}_v$ holds for all orders: first, it is placed by the customer (t^{avl}_v), then it is picked from the warehouse (t^{rel}_v) and ready for delivery by a driver, and then due (t^{due}_v). Note that orders that are placed later may be due earlier. The earliest route departure time is bound from below by the latest release time of the corresponding orders. For this particular example $\tau_r \geq t^{\text{rel}}_{v_3}$ must hold.

5.3 Problem Formalization

We distinguish between three problem variants: the offline problem with full knowledge of the day in advance (OFF), the dynamic problem at a specific time $\tilde{t}$ (DYN-$\tilde{t}$), and the full dynamic problem for a whole day (DYN-DAY).

5.3.1 Full-Knowledge Offline Problem (OFF)

Here all orders of the day are known in advance together with their release times, i.e., the times the orders have been picked in the warehouse and are ready for delivery by the drivers. Although this problem variant is not what we are confronted with in reality, it is nevertheless interesting as its (optimal) solution provides a baseline of what might ideally be achieved in the online problem. We denote the set of all orders by $V = \{1, \ldots, n\}$ and the corresponding release times by t^{rel}_v, $\forall v \in V$. Moreover, we are given due times t^{due}_v, $\forall v \in V$, which are related to a promised maximum delivery duration starting from the time t^{avl}_v the order v was placed by the customer. Fig. 5.1 visualizes the order-related times of an example route consisting of three orders.

Furthermore, for all relevant vehicles $u \in U$, with $m = |U|$, planned shift time intervals $[q^{\text{start}}_u, q^{\text{end}}_u]$ and earliest shift ends $q^0_u \in [q^{\text{start}}_u, q^{\text{end}}_u]$ are provided. Lastly, expected travel times $\delta(v, v')$ from location v to location v', where $v, v' \in V \cup \{0\}$ with 0 representing the warehouse, are given. These travel times include average stop times at the customers, average times for loading a vehicle at the warehouse, and postprocessing times when returning to the warehouse. We further assume that the triangle inequality holds w.r.t. the travel times and that they are constant throughout the day.

Solution Representation. We have to plan the drivers' routes, the route departure times, and the drivers' flexible shift end times. Hence, a candidate solution is a tuple $\langle R, \tau, q \rangle$ where

- $R = (R_u)_{u \in U}$ denotes the *ordered sequence of routes* $R_u = \{r_{u,1}, \ldots, r_{u,\ell_u}\}$ to be performed by each vehicle $u \in U$, and each *route* $r \in R_u$ is an ordered sequence $r = \{v^r_0 = 0, v^r_1, \ldots, v^r_{l_r}, v^r_{l_r+1} = 0\}$ with $v^r_i \in V$, $i = 1, \ldots, l_r$, being the i-th order to be delivered and 0 representing the warehouse at which each tour starts and ends,
- $\tau = (\tau_r)_{r \in R_u, u \in U}$ are the *departure times* of the routes, and

- $q = (q_u)_{u \in U}$ are the *shift end times* of the vehicles.

The time at which the i-th order v_i^r of route r, $i = 1, \ldots, l_r$, is delivered is

$$a(r, i) = \tau_r + \sum_{j=0}^{i-1} \delta(v_j^r, v_{j+1}^r). \tag{5.1}$$

The total duration of a route $r \in R_u$ of a vehicle $u \in U$ is

$$d(r) = \sum_{i=0}^{l_r} \delta(v_i^r, v_{i+1}^r), \tag{5.2}$$

and the route, therefore, is supposed to end at time $\tau_r + d(r)$.

Let $\tau^{\min}(r) = \max_{i=1,\ldots,l_r} t^{\text{rel}}_{v_i^r}$ be the earliest feasible starting time of a route r, which corresponds to the maximum release time of the orders served in the route. Furthermore, let $\tau^{\max}(r)$ be the latest starting time without violating any due time, i.e.,

$$\tau^{\max}(r) = \min_{i=1,\ldots,l_r} \left(t^{\text{due}}_{v_i^r} - \sum_{j=0}^{i-1} \delta(v_j^r, v_{j+1}^r) \right). \tag{5.3}$$

Feasibility. A solution is feasible when

- each order $v \in V$ appears exactly once in all the routes in $\bigcup_{u \in U} R_u$,
- each route $r \in R_u$, $u \in U$, is started in the planned shift time of the assigned vehicle, i.e., $\tau_r \in [q_u^{\text{start}}, q_u^{\text{end}}]$,
- and not started before all corresponding orders are released, i.e., $\tau_r \geq \tau^{\min}(r)$,
- the routes in each R_u, $u \in U$ start at increasing times and do not overlap, i.e., $\tau_{r_{u,i}} + d(r_{u,i}) \leq \tau_{r_{u,i+1}}$, $i = 1, \ldots, |R_u| - 1$,
- and the actual shift end time is not smaller than the finishing time of the last route (if there is one) and the minimum shift time, i.e., $q_u \geq \max(q_u^0, \sup_{r \in R_u}(\tau_r + d(r)))$, $u \in U$.

Objective. The primary goal is to avoid tardiness or distribute it evenly among the customers. The secondary goal is to reduce labor and travel costs. This leads to the following objective function to be minimized

$$f(\langle R, \tau, q \rangle) = \mathrm{L}\left(\sum_{r \in R_u, u \in U} \sum_{i=1}^{l_r} \max(0, a(r, i) - t^{\text{due}}_{v_i^r})^2,\ \gamma \cdot \sum_{u \in U} (q_u - q_u^0) + \sum_{r \in R_u, u \in U} d(r) \right). \tag{5.4}$$

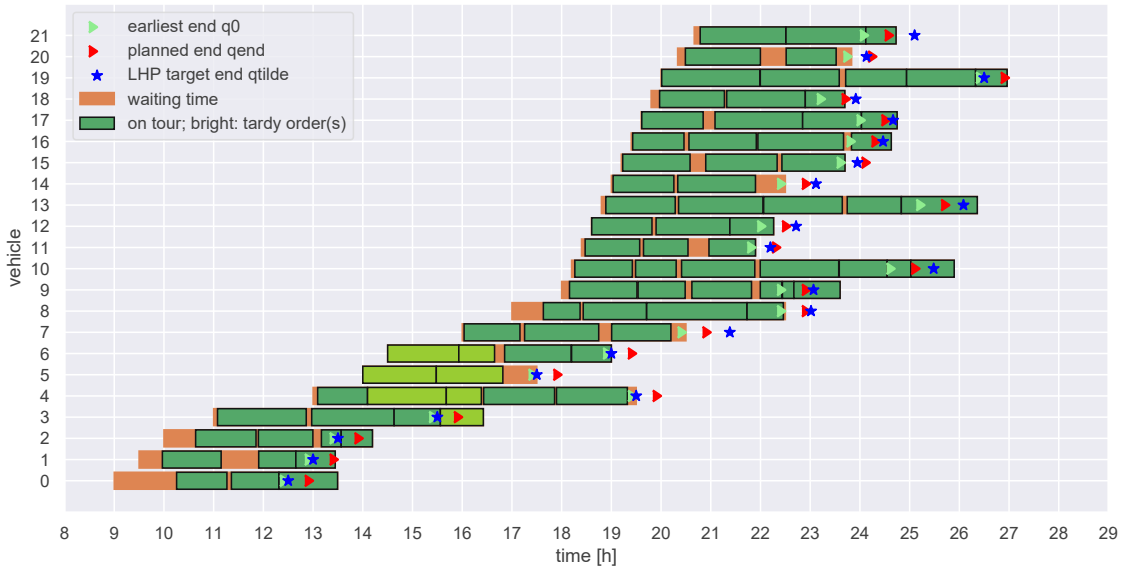


Figure 5.2: Visualization of a solution for an artificial instance with 22 vehicles. The x-axis denotes the time and the discrete y-axis the drivers' shifts. The whole bar indicates the actual shift duration. The green triangle indicates the earliest shift end time for each driver, where excess labor time contributes to our considered costs. The red triangle depicts the planned shift ending, after which no route can be started, but the last route may end arbitrarily late. The distinct green bars stand for routes and contribute to the travel time part of our objective function. If the latter is shaded light green, the route contains at least one tardy order, which can be observed around hour 15. The remaining orange of the shift bars denotes waiting time of the driver at the depot. The blue stars denote the target end shift times as determined by the large horizon planning at the beginning of the day.

L denotes the lexicographic combination of two terms, which are a quadratic penalty for the tardiness of deliveries and a linear combination of the sum of labor and travel costs. More precisely, the latter is calculated as the sum of the actual shift durations above the minimum shift times q_u^0 weighted by a factor γ and the sum of travel times.

In a real-world comparison of results, it is also worthwhile to view it as a multi-objective optimization problem. A small increase in tardiness may be acceptable, if it comes with a substantial reduction of costs.

5.3.2 Dynamic Problem at a Specific Time $\tilde{t}$ (DYN-$\tilde{t}$)

This problem variant is actually the one that needs to be iteratively solved during the whole day, for increasing current time $\tilde{t}$. It extends OFF by having as additional input the current time $\tilde{t}$ and the expected number of orders $\hat{\omega}(t)$ that become available in the time intervals $[t, t + 1\text{h})$ for all relevant business hours. Moreover, we assume knowledge about the distribution of order types w.r.t. the promised delivery durations. The set of all orders V is reduced to those which are already available at time $\tilde{t}$ and whose delivery has not yet started. The set of vehicles U is reduced to those whose shift has not been finished, and shift start times are updated to the expected return times of vehicles that are currently on a tour. The route construction must now additionally consider these unknown future orders in an appropriate way. The ultimate goal is to lead to an optimal

solution w.r.t. the full dynamic problem below.

5.3.3 Full Dynamic Problem (DYN-DAY)

This is the actual problem to be solved from the point-of-view of the whole day. Time is considered to continuously increase over the whole relevant time horizon, expected numbers of future orders are known as above, but each concrete order becomes available only at the availability time t_v^{avl}, $\forall v \in V$. The decision on each route $r \in R$ must be fixed with only the knowledge available up to the route's respective departure time τ_r. An example solution of a DYN-DAY instance with 22 vehicles is depicted in Fig. 5.2 as a bar chart displaying the waiting times (orange), routes (green), and routes with tardy orders (light green) of the drivers. Stars show the target shift end times derived from our initial large horizon planning (Sect. 5.7).

More specifically, we solve the successive DYN-$\tilde{t}$ instances every time an order is released:

$$\tilde{t} \in \left\{ t \mid \exists v \in V : t = t_v^{\text{rel}} \right\} \tag{5.5}$$

Having obtained a solution for a time $\tilde{t}$, we extract any routes that start before the next value for $\tilde{t}$ in the above sequence, adopt these routes for the final solution of DYN-DAY, and remove all the orders served in these routes from any further consideration.

5.4 Routes Construction and Optimization

To be suitable for a real-time application, an important property that an optimization method must exhibit is a good *anytime behavior*: a somehow reasonable heuristic solution must be found very soon (within seconds), and over time the solution should continuously be improved up to (or close to) optimality. In other words, the optimization can be interrupted almost at any time and a reasonable solution with respect to the invested time is available. We achieve this by using a carefully designed ALNS [116, 127].

ALNS heuristics. As construction heuristics to insert orders into either an empty solution or to repair a partial solution in the ALNS, we use the well-known insertion and regret-k heuristics as described in [117]. We distinguish between the *zero-tardiness* and *tardiness* regimes. In a two-stage approach, we first seek to insert an order without introducing additional tardiness, which can be checked in constant time with caching of suitable slack values for existing orders and routes. If this is not possible, we search for an order position with the smallest sum of squares increase of tardiness, which is computationally more demanding by a factor of $\mathcal{O}(n)$.

Our destruction heuristics are mostly adopted from Pisinger and Ropke [117], Ropke and Pisinger [127], Shaw [135], and Azi et al. [8] and suitably adapted to our problem. There are two kinds of destruction heuristics, those that remove a certain number of orders from routes and those that remove a certain number of whole routes. More specifically,

we use *random order*, *random route*, *related order*, *related route*, *worst order*, and *worst and related order* removal.

Shift End Times. The vehicles' actual shift end times q_u are set to $\max(q_u^0, \sup_{r \in R_u}(\tau_r + d(r)))$, i.e., for each vehicle u to the end of the last route or the earliest possible shift end time, whichever comes later. In Section 5.7, we introduce the large horizon planning, where we estimate desired shift ends for the vehicles in advance so that we can satisfy the expected workload. Since in the objective function we penalize labor time after the earliest possible shift end times q^0, we grant vehicles that are below their desired shift end time $\tilde{q}_u > q_u^0$ a labor time bonus that equalizes the incurred labor time costs up until $\tilde{q}_u$—otherwise, the insertion heuristic would avoid assigning orders to vehicles after their earliest possible shift end q_u^0, in case there is no tardiness yet and other vehicles not close to their shift end are available. The labor time bonus is implemented by using the augmented objective function

$$\tilde{f}(\langle R, \tau, q \rangle)) = f(\langle R, \tau, q \rangle) - \gamma \cdot \sum_{u \in U} \min(q_u - q_u^0, \tilde{q}_u - q_u^0). \tag{5.6}$$

During the optimization, the route departure time is always set to the earliest possible time. Afterwards, we are free to postpone the routes up to the latest time within the departure time slacks of the routes so that the objective value is neither increased by tardiness nor by labor costs.

5.5 Driver Performance Estimation

For both an informed route departure time strategy and our large horizon approach, we need to estimate the driver performance of a given hour. It is the average time needed to serve an order. It is strongly related to the expected duration of all routes involved to serve the customers at the considered time interval divided by the number of customers. We introduce this as a function $\phi\colon \mathbb{R} \to \mathbb{R}$, depending on the load λ. We define the load λ to be the expected number of orders due in a given hour.

A classical result by Beardwood et al. [9] shows that the expected length of an optimal traveling salesperson tour with n randomly sampled cities given some geometry with area $\mathcal{A}$ grows with $k\sqrt{\mathcal{A}n}$. k is an empirical constant depending on the spatial distribution and the metric. This result is extended to capacitated vehicle routing problems by Daganzo [35] and refined by Figliozzi [51], from which we adapt the following model

$$\phi(\lambda; k_m, k_l) \approx k_m + \frac{k_l}{\sqrt{\lambda + 1}}. \tag{5.7}$$

k_m corresponds to constant costs occurring for each customer like the stop time at the customer. k_l relates to the empirical k from [9] and accounts together with $(\lambda + 1)^{-1/2}$ for the expected travel time to a customer. We shift the load by one to avoid divergence

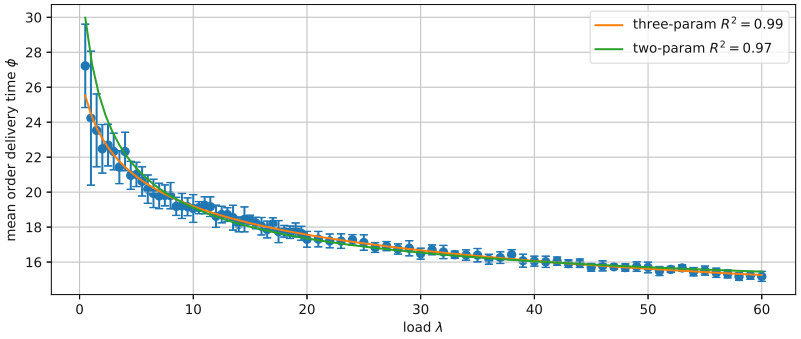


Figure 5.3: Mean order delivery times ϕ in minutes with standard errors over ten instances for each load value $\lambda \in \{0.5, 1.0, \ldots, 20, 21, \ldots, 60\}$ with fitted curves for the two and three-parameter models. The three-parameter model seems to explain the region of little load $\lambda \leq 10$ better than the two-parameter model.

at zero load. As we can see, it is a function that decreases with the square root of the load. As a more flexible model, we further suggest the following inverse power law

$$\phi(\lambda; k_m, k_l, \alpha) \approx k_m + \frac{k_l}{(\lambda + 1)^\alpha}. \tag{5.8}$$

To check the validity of these models in our setting and tune the parameters, we create ten artificial instances each for loads starting from 0.5 up to 20 in steps of 0.5 and further in steps of 1 up to 60, i.e., to one order due per minute. The geometry is the unit disk with a central depot, Euclidean metric, and vehicles driving a constant pace of 20 minutes per unit distance. Furthermore, constant stop times at the customers, loading times when leaving the warehouse, and postprocessing times when returning to the warehouse are added. Orders arrive randomly throughout a whole day at a given constant rate λ sampled from a Poisson process following a uniform spatial distribution. Optimization at each DYN-$\tilde{t}$ is done for 60 seconds using ALNS. Sufficient drivers are available so that no tardiness occurs, and the drivers wait to start their routes as long as possible. For each instance, we average over all routes the time needed to serve a customer.

In Fig. B.2 we see a scatter plot of the mean order delivery times including standard errors ($N = 10$) over the different loads. Weighted least squares fits of the models are displayed. Both models explain the data starting from load $\lambda \geq 10$ similarly well with weighted R^2 values of 0.97 and 0.99. For low-load regions $\lambda \leq 10$, the model with the inverse power as an arbitrary parameter lies closer to the means.

5.6 Departure Time Strategies

In the dynamic problem, at every time $\tilde{t}$ we construct and optimize the routes for the drivers. In a decomposition approach, we decide afters separately when these should be started, i.e., the scheduling of the routes. A departure time window $[\tau_r^{\text{earliest}}, \tau_r^{\text{latest}}]$ is attributed to each route, within which the departure time τ_r of the route may be set while maintaining a feasible solution and not increasing the objective value. Setting

$\tau_r < \tau_r^{\text{earliest}}$ or $\tau_r > \tau_r^{\text{latest}}$ makes the solution either infeasible or increases tardiness, labor, or travel costs. This decision is crucial since routes cannot be adapted anymore after they have been started.

Two extreme strategies can immediately be devised by either always starting the route at τ^{earliest} or at τ^{latest}, see, e.g., [109]. The latter, τ^{latest}, seems favorable in terms of route quality since not yet started routes may later still be adapted in order to more efficiently include newly emerged orders as opposed to the earliest strategy where in the extreme case a route is immediately started with just one order. However, experiments have shown that the start-latest strategy is not always the better strategy, since we may run into tardiness at a later time when working at or shortly before critical utilization and letting vehicles wait instead of delivering orders.

A more informed heuristic approach takes into account the current performance of a route, measured by its number of minutes per order d_r^{O}, i.e., the route duration divided by the number of orders served. The main idea is: the better the performance of a route, the closer we can set its departure time τ_r towards τ_r^{earliest}, the worse, the closer towards τ_r^{latest}, so that there is a performance-dependent time for improvement by further incoming orders. As we have seen in detail in Section 5.5, the performance depends on the load by an inverse power law.

We assume a Gaussian distribution of $d_r^{\text{O}} \sim \mathcal{N}(\phi(\lambda), \sigma_\phi^2(\lambda))$ and set the departure time of a route to

$$\tau_r(d_r^{\text{O}}, \lambda) = \tau_r^{\text{earliest}} + (\tau_r^{\text{latest}} - \tau_r^{\text{earliest}}) \cdot \Phi\left(\frac{d_r^{\text{O}} - \phi(\lambda)}{\sigma_\phi(\lambda)}\right), \tag{5.9}$$

where Φ is the cumulative normal distribution function. For example, when d_r^{O} corresponds exactly to the expected mean order delivery time $\phi(\lambda)$ in the given load situation, the departure time of r will be set to $(\tau_r^{\text{earliest}} + \tau_r^{\text{latest}})/2$, the middle of the route departure time slack. We estimate $\sigma_\phi(\lambda)$ by calculating sample standard deviations from our experiments described in the previous section. An illustration of this strategy is shown in Figure 5.4.

We will refer to the three different strategies as τ-earliest, τ-latest, and τ-route.

5.7 Double Horizon Approach

This approach adopts from Mitrović-Minić et al. [109] the idea of considering in the optimization two planning horizons simultaneously, a short horizon and a large horizon. In the large horizon planning (LHP), which we always perform as the first step, we consider a strongly simplified approximate problem variant of DYN-$\tilde{t}$ where, in addition to all available requests, also all the expected future requests for either the whole day or at least several hours into the future. The primary goal is to make a rough plan for the utilization of the vehicles and recognize times when we might exceed the available capacity or have enough time to finish vehicle shifts earlier. A detailed routing is *not*

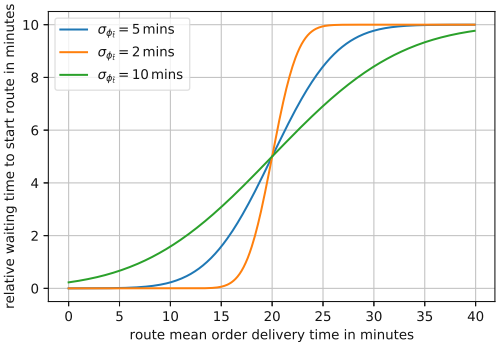


Figure 5.4: Illustration of Gaussian switch strategy with different variances controlling the smoothness of the transition. The more efficient a route relative to the mean performance is, the earlier it is started, and vice versa.

done in the LHP. The short horizon problem corresponds to our definition of DYN-$\tilde{t}$ so far but utilizes an adapted objective function that includes additional terms defined by the LHP's results in order to meet the long-term goals as closely as possible. In our case decisions on the labor time to be used beyond the minimum q_u^0 for each vehicle are most critical in the long term in order to avoid later deliveries becoming tardy due to insufficient driving resources for the given workload.

We, therefore, define and solve the following LHP subproblem at time $\tilde{t}$ in order to derive *target shift end times* $\tilde{q}_u$ for each vehicle $u \in U$. We consider as V all currently relevant orders of the current DYN-$\tilde{t}$ plus expected orders V^{exp} for the remaining day. These expected orders are artificially created according to the estimated numbers of orders becoming available per hour $\hat{\omega}(t)$, equidistantly spaced over each hour. For each of these orders, we further derive a due time randomly based on the distribution of expected order types and their promised maximum delivery times.

Let $z\colon \mathbb{R}^+ \to \mathbb{R}^+$ be a function that estimates the average shift duration needed to serve one order $v \in V \cup V^{\text{exp}}$ within the current hour of $\tilde{t}$ and a few subsequent hours, assuming a reasonable routing and an average number of available orders. The basis for z is the mean order delivery time $\phi(\lambda)$ derived from the routes with latest departure time strategy as presented in Section 5.5. To account for a slight increase due to waiting times in the depot and an intermediate departure time strategy, we introduce an additional factor $\zeta \gtrapprox 1$, which needs to be tuned. With $\Lambda(t)$ being the load at hour t, we then calculate a weighted average to estimate the average shift duration

$$z(\tilde{t}) = \zeta \cdot \frac{\sum_{t'=\tilde{t}}^{\tilde{t}+\rho} \Lambda(t') \cdot \phi(\Lambda(t'))}{\sum_{t'=\tilde{t}}^{\tilde{t}+\rho} \Lambda(t')}, \tag{5.10}$$

with ρ corresponding to three hours in our implementation. We make the strongly simplifying assumption that any order v can be independently served by any available vehicle within time $z(\tilde{t})$ from t_v^{rel} onward. Each vehicle's shift is split into successive time slots of duration $z(\tilde{t})$, and in each of these time slots, one order can be served. This

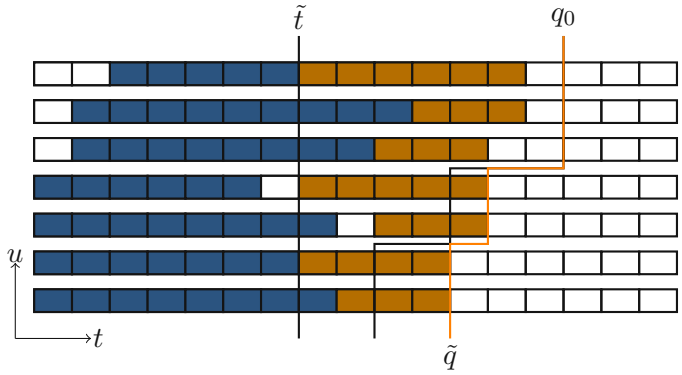


Figure 5.5: View on an example DYN-$\tilde{t}$ problem instance as seen by the large horizon planning. The x-axis represents the time, discretized by time slots of the average expected shift time needed to deliver an order $z(\tilde{t})$. The drivers that are still available at or after $\tilde{t}$ are stacked on the y-axis. Blue rectangles indicate orders that have already been delivered or are en route. Brown rectangles represent greedily assigned orders, either real (available at the moment) or expected up until the end of the planning horizon. The maximum of the earliest shift end time q_u^0 and the latest assigned order define for each driver the target shift end time $\tilde{q}_u$. For the last four drivers, q_0 is exceeded, since otherwise, tardiness would have arisen. Note that unassigned slots may occur if no more orders are ready for delivery at that time.

implies that we do not allow arbitrary start times to serve orders but only times that are multiples of $z(\tilde{t})$ away from a vehicle's shift start time (or $\tilde{t}$). We do not have a strict last slot, i.e., in principle further orders to be served might always be appended to a vehicle's shift. An instructive visualization of the LHP's view on an example DYN-$\tilde{t}$ instance is provided in Figure 5.5.

A solution to our LHP is a complete assignment of all the orders $V \cup V^{\text{exp}}$ to vehicle slots. As actual delivery time of an order, we consider the respective time slot's middle point, i.e., the time slot's start time plus $z(\tilde{t})/2$. The objective function corresponds to our main objective function (5.4), but as we do not consider routing the last travel time term is omitted.

This LHP is heuristically solved by a greedy assignment procedure, in which orders are assigned in increasing due time always to the earliest feasible time slot of a vehicle that increases the objective the least. In case of ties, a vehicle $u \in U$ whose end of the shift exceeds q_u^0 the least, i.e., where the vehicle's excess labor time is smallest, is chosen. This aspect automatically balances the deviations from the planned shift times among the vehicles if there are no particular other reasons such as avoiding tardy orders. Further ties are resolved randomly by a random processing order of the vehicles.

The obtained shift end times of this solution, i.e., the end times of the last used time slots of each vehicle, are finally used as target shift end times $\tilde{q}_u$, for all $u \in U$ in the short horizon optimization, i.e., the ALNS from Section 5.4.

This is achieved by augmenting objective function (5.4) to

$$\tilde{f}(\langle R, \tau, q \rangle)) = f(\langle R, \tau, q \rangle) + \gamma \cdot \sum_{u \in U} Q_u \tag{5.11}$$

with

$$Q_u = -\min(q_u - q_u^0, \tilde{q}_u - q_u^0). \tag{5.12}$$

This non-positive term can be seen as a bonus that exactly compensates any arising labor time costs above q_u^0 up to the target time $\tilde{q}_u$ for each vehicle $u \in U$. Thus, the time up to $\tilde{q}_u$ can be used "for free". Note that the factor γ by which the bonus is multiplied is the same as by which the labor time is weighted in (5.4).

5.8 Shift Ending Strategies

In the online problem, we also have to decide if a shift should be ended by sending a driver (vehicle) $u \in U$ home, providing this is allowed, i.e., $\tilde{t} \geq q_u^0$, u is in the depot, and no more routes are planned for u, or if the driver has to wait at the depot to possibly receive further orders. Again, two naive strategies are immediately available: The first option is to send a vehicle u home as early as possible, i.e., after its last so far planned route or at q_u^0, whichever comes later. This is also the default of the insertion heuristic. The other extreme is the latest strategy that waits until q_u^{end} in any case, even if the last route ends before q_u^{end}. The earliest strategy seems to be an attractive choice since we can save labor costs and during peak hours, it is likely that a vehicle has already a next route planned during its current route, therefore it is not sent home prematurely when arriving at the depot if there is still enough work to do.

A more sophisticated approach makes use of the estimated shift end times $\tilde{q}$ provided by the LHP. The earliest shift end time is then modified to be $\tilde{q}_u - \tilde{d}$, where $\tilde{d}$ is a threshold duration of an efficient route. The rationale is that if a vehicle cannot start a somewhat efficient route that ends before its target shift end $\tilde{q}_u$, it is better to send it home.

We will refer to the three different strategies as q-earliest, q-latest, and q-LHP.

5.9 Computational Study

We conducted all our experiments on Intel Xeon E5-2640 processors with 2.40 GHz in single-threaded mode and a memory limit of 8 GB. We implemented our approach as a prototype in Python 3.7, being aware that an implementation in a compiled language would be substantially faster and have a smaller memory footprint. We consider six different instance classes, each with 20 instances: Artificial instances[1] on the Euclidean unit disk as described in Section 5.5 using either a business day (BD) or a weekend (WE) load profile with generous (GE), tight shift planning (TI), and with a shortage (SH) of drivers. The instance generation algorithm is described in detail in Appendix B.1. The idea is to observe the transition from a more generous shift planning to a tighter one and to simulate a driver being absent on short notice where in the latter cases more tardiness is expected to occur. Furthermore, in the generous case, dynamically ending shifts earlier

[1] https://github.com/nfrohner/pdsvrpddsf

Algorithm 5.1: Simulated DYN Problem Solver with ALNS and LHP.

Input: Orders V, drivers U, shift starts q^{start}, earliest shift ends q^0, planned shift ends q^{end}, hourly expected number of orders $\hat{\omega}$, travel time matrix δ, mean order delivery time ϕ, efficient route threshold duration $\tilde{d}$.

Output: Solution $\langle R, \tau, q \rangle$ with routes R, route departure times τ, and actual shift end times q for the whole day.

$V^{\text{deliv}} \leftarrow \{\}$, $U^{\text{home}} \leftarrow \{\}$;
$R' \leftarrow ()_{u \in U}, \tau' \leftarrow ()_{r \in R'}, \tilde{t}' \leftarrow 0, \tilde{q} \leftarrow q^{\text{end}}$;
$\langle R, \tau, q \rangle \leftarrow \langle R', \tau', q^0 \rangle$;
foreach $\tilde{t} \in \left\{ t \mid \exists v \in V : t = t_v^{\text{rel}} \right\} \cup \{\infty\}$ **do**
  foreach $(u, r) \in R' : \tilde{t}' \leq \tau'_r < \tilde{t}$ **do**
    $V^{\text{deliv}} \leftarrow V^{\text{deliv}} \cup \{v_1^r, \ldots, v_{l_r}^r\}$;
    $\langle R, \tau, q \rangle \leftarrow \langle R, \tau, q \rangle \oplus (r, \tau'_r)$;
    $q_u^{\text{start}} \leftarrow u$'s return time at depot after $\tilde{t}$;
  end
  $U^{\text{home}}, q \leftarrow \text{SENDHOME}(\tilde{t}, \tilde{t}', U^{\text{home}}, U \setminus U^{\text{home}}, q^{\text{start}}, q^0, q^{\text{end}}, \tilde{q}, \tilde{d})$;
  $V^{\text{avl}} \leftarrow \{v \in V \setminus V^{\text{deliv}} : t_v^{\text{avl}} \leq \tilde{t}\}$;
  $\tilde{q} \leftarrow \text{LHP}(\tilde{t}, V^{\text{avl}}, U \setminus U^{\text{home}}, \hat{\omega}, \phi, q^{\text{start}}, q^0, q^{\text{end}})$;
  $\langle R', \tau', q \rangle \leftarrow \text{ALNS}(\tilde{t}, V^{\text{avl}}, U \setminus U^{\text{home}}, q^{\text{start}}, q^0, q^{\text{end}}, \tilde{q}, \delta)$;
  $\tau' \leftarrow \text{DEPART}(R', \phi)$;
  $\tilde{t}' \leftarrow \tilde{t}$;
end
return $\langle R, \tau, q \rangle$;

is expected to have more impact whereas in the tight case, shifts are more likely to be extended by starting long routes shortly before the ending.

We aim at comparing the performance of the naive earliest and latest strategies with the more sophisticated LHP and driver performance-based route departure strategy on these DYN problem instances. In each case, we apply the ALNS with a limit of 1000 non-improving iterations and additionally a 60 seconds time limit for route optimization at each arriving order. This should be consistent with a real-time setting, where orders may arrive every minute during peak time or on weekends and routes already including them should occasionally be started within a minute. Without LHP and driver performance estimation, we are restricted to naive earliest and latest strategies regarding the departure time of a route and the early shift termination. LHP extrapolates until the end of the long horizon to set desired shift ending times for each driver, using an estimation of the average driver performance in the window of the current and the upcoming three hours. The target shift ending times may be before the planned shift ends to send drivers home early or after them so that extending shifts is favored via the augmented objective function.

Table 5.1: Offline problem performance (OFF) and different solution strategies applied to 20 artificial instances for each configuration using either a business day (BD) load profile with either generous, tight, or shift planning with a driver shortage.

			n^{tardy}		RMSE [min]		dur [h]		lab [h]		$\bar{d}$ [min]		d_r^{O} [min]	
			mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
load	shift	solving strategy												
BD	generous	τ-earliest, q-earliest	4.500	4.536	0.680	0.798	98.134	3.580	1.491	1.069	56.425	4.315	18.386	0.692
		τ-earliest, q-latest	3.850	4.771	0.592	0.795	98.087	3.984	11.940	0.698	55.930	3.707	18.372	0.631
		τ-latest, q-LHP	6.250	6.257	1.107	1.208	88.221	4.447	2.685	1.199	74.935	2.832	16.512	0.426
		τ-latest, q-earliest	7.600	6.065	1.157	1.091	87.716	4.443	1.673	0.889	74.745	2.818	16.420	0.479
		τ-latest, q-latest	5.900	5.937	1.028	1.237	87.980	5.210	11.974	0.722	75.460	2.792	16.463	0.574
		τ-route, q-LHP	4.400	4.604	0.841	1.134	89.369	4.568	2.197	1.560	71.340	2.113	16.728	0.483
		τ-route, q-latest	4.350	4.782	0.757	1.082	89.956	4.241	12.036	0.797	70.060	2.398	16.839	0.379
		OFF	0.550	1.572	0.056	0.174	78.897	4.763	0.377	0.479	64.170	2.039	14.760	0.384
	shortage	τ-earliest, q-earliest	19.950	16.804	3.991	4.966	91.863	4.294	8.334	3.843	63.415	4.985	17.259	0.468
		τ-earliest, q-latest	19.100	13.726	3.381	2.568	92.763	4.305	14.775	2.426	62.930	4.941	17.430	0.533
		τ-latest, q-LHP	14.800	10.928	2.705	2.915	88.099	4.742	12.262	3.854	74.640	3.749	16.544	0.350
		τ-latest, q-earliest	26.200	14.207	4.179	2.747	87.507	5.044	9.946	4.296	74.355	4.007	16.430	0.370
		τ-latest, q-latest	22.737	16.562	3.551	2.820	87.524	5.038	15.406	2.073	72.126	3.019	16.505	0.426
		τ-route, q-LHP	15.450	11.019	2.546	2.525	89.483	4.717	11.454	4.103	71.270	3.763	16.806	0.388
		τ-route, q-latest	18.150	11.864	2.988	2.338	89.992	4.691	16.304	2.145	70.045	3.622	16.903	0.375
		OFF	2.050	5.826	0.255	0.721	81.655	4.358	4.671	3.230	57.595	1.676	15.336	0.400
	tight	τ-earliest, q-earliest	14.158	10.673	2.720	2.928	93.582	3.530	6.319	2.803	62.416	4.261	17.507	0.746
		τ-earliest, q-latest	10.950	8.003	2.847	4.480	94.496	3.263	14.236	1.601	61.630	3.886	17.670	0.725
		τ-latest, q-LHP	14.300	11.855	2.139	1.923	89.265	5.296	9.496	3.458	73.180	2.846	16.667	0.477
		τ-latest, q-earliest	17.150	9.544	2.959	2.341	88.868	5.074	7.920	2.934	72.865	2.211	16.593	0.457
		τ-latest, q-latest	13.900	9.754	2.466	2.179	88.732	4.822	14.810	1.636	70.945	2.330	16.571	0.514
		τ-route, q-LHP	8.950	6.117	1.843	2.098	90.030	4.238	8.883	2.960	70.370	2.580	16.818	0.452
		τ-route, q-latest	10.105	7.880	2.028	2.542	90.265	4.403	14.728	1.231	68.816	1.937	16.870	0.509
		OFF	0.700	2.904	0.115	0.512	82.371	5.380	3.273	2.645	58.400	2.307	15.374	0.420

In Algorithm 5.1 we list a high-level pseudo-code of the simulated DYN problem solver, combining the previously explained approaches based on the LHP and route performance. The main loop goes over all times $\tilde{t}$ where an order is released, where the first inner loop checks whether routes have been started between the last and the current $\tilde{t}$. If so, they are added to the current solution, the corresponding orders are removed, and the drivers' shift starts are set to their return times at the depot. Afterwards, drivers are sent home, if their target shift end time reduced by the efficient route threshold $\tilde{q} - \tilde{d}$ passed and they have no further routes planned. Then the route construction and optimization begins with the large horizon planning to update the $\tilde{q}$. It further continues with the ALNS—the optimization workhorse—that creates routes for the currently available and not yet delivered orders. Finally, the departure time of the planned routes is set according to the route performance strategy (more efficient routes are planned to start earlier).

In Table 5.1 and 5.2, we present the main results of our computational study. We

Table 5.2: Offline problem performance (OFF) and different solution strategies applied to 20 artificial instances for each configuration using a weekend (WE) load profile with either generous, tight, or shift planning with a driver shortage.

			n^{tardy}		RMSE [min]		dur [h]		lab [h]		$\bar{d}$ [min]		d_r^{O} [min]	
			mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
load	shift	solving strategy												
WE	generous	τ-earliest, q-earliest	2.400	3.267	0.420	0.539	145.071	3.794	1.213	1.660	62.140	3.102	17.008	0.504
		τ-earliest, q-latest	1.750	2.900	0.273	0.435	145.130	3.561	15.889	0.853	61.985	3.286	17.021	0.634
		τ-latest, q-LHP	2.850	2.661	0.701	1.056	132.071	4.851	2.024	1.885	76.835	2.059	15.477	0.343
		τ-latest, q-earliest	2.850	2.978	0.745	1.112	132.159	5.094	1.715	1.753	76.580	2.008	15.486	0.334
		τ-latest, q-latest	1.600	2.137	0.358	0.531	132.161	4.962	16.099	0.859	76.015	1.727	15.488	0.335
		τ-route, q-LHP	2.450	2.837	0.455	0.551	134.841	4.713	1.820	1.657	73.420	2.295	15.800	0.329
		τ-route, q-latest	1.650	2.641	0.271	0.411	134.682	4.460	16.286	1.248	73.945	1.934	15.787	0.435
		OFF	0.000	0.000	0.000	0.000	118.698	5.720	0.197	0.407	67.455	2.253	13.902	0.291
	shortage	τ-earliest, q-earliest	12.950	9.801	1.743	1.817	139.469	3.321	9.636	2.564	67.160	3.392	16.167	0.597
		τ-earliest, q-latest	7.400	5.623	1.475	2.305	139.888	3.335	19.844	1.440	66.135	2.164	16.215	0.571
		τ-latest, q-LHP	9.900	7.440	1.077	0.923	132.993	3.421	12.833	2.534	77.185	1.991	15.411	0.350
		τ-latest, q-earliest	14.250	9.107	1.527	1.578	132.510	3.574	11.427	2.400	75.295	1.975	15.354	0.380
		τ-latest, q-latest	11.700	9.985	1.632	1.878	132.087	3.426	20.397	1.485	75.850	2.883	15.305	0.320
		τ-route, q-LHP	8.100	8.534	0.924	1.048	135.573	3.432	12.724	3.585	74.550	1.673	15.711	0.413
		τ-route, q-latest	7.400	8.068	1.042	0.898	136.989	3.011	22.970	1.615	73.765	1.808	15.875	0.371
		OFF	0.350	0.988	0.032	0.130	123.384	3.617	4.860	2.077	61.880	1.426	14.296	0.330
	tight	τ-earliest, q-earliest	12.200	16.938	2.372	3.017	141.820	3.805	8.029	3.336	64.295	3.578	16.608	0.526
		τ-earliest, q-latest	8.350	7.707	1.558	1.924	142.144	4.282	19.164	1.851	64.055	3.246	16.643	0.528
		τ-latest, q-LHP	7.050	8.587	1.121	1.442	133.109	5.952	11.178	4.118	76.335	2.057	15.575	0.261
		τ-latest, q-earliest	10.850	10.246	2.109	2.531	132.394	5.840	9.369	3.762	75.115	1.923	15.490	0.285
		τ-latest, q-latest	8.500	8.237	1.955	2.667	132.156	5.971	19.720	1.794	73.955	2.546	15.462	0.311
		τ-route, q-LHP	6.050	6.739	1.113	1.610	135.414	5.522	10.216	4.080	73.805	1.910	15.848	0.365
		τ-route, q-latest	5.050	6.700	1.021	1.536	136.518	4.566	21.582	1.895	73.285	2.089	15.980	0.347
		OFF	0.800	2.118	0.079	0.189	122.279	6.193	3.716	2.530	62.540	1.653	14.303	0.285

compare for the different combinations of our approaches means and standard deviations of the number of tardy orders n^{tardy}, the root mean squared error (RMSE) of the tardiness in minutes, the total travel time in hours, the labor time exceeding q_u^0 in hours, the average route duration $\bar{d}$, and the average route performance (labor time to serve an order without waiting time) in minutes d_r^{O}. The offline (full knowledge of the day) results (OFF) where we applied ALNS to convergence with a limit of 1000 non-improving iterations without additional time limit provide a performance baseline. All the other results are for the DYN-DAY problem variant and we see that the offline baseline is somewhat out of reach, which is not too surprising due to the substantially restricted knowledge that can be exploited in the online variant. Despite having used a lexicographic optimization approach, where distributing tardiness evenly and reducing it was the single most important objective, we analyze the results in the sense of a multi-objective optimization problem. Small amounts of tardiness for a few customers

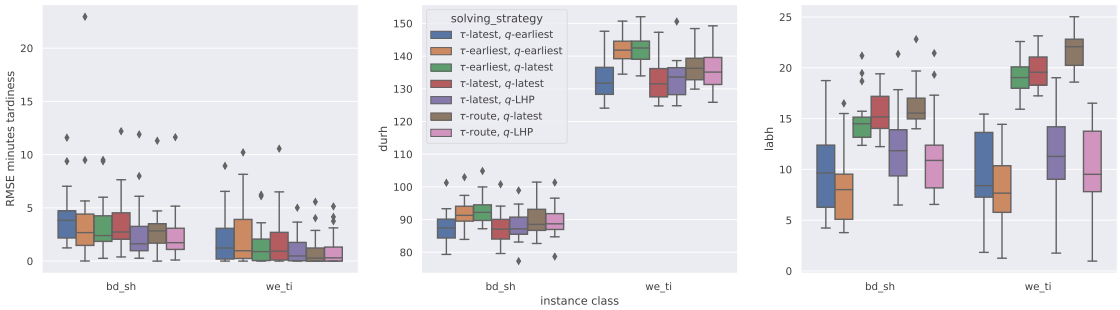


Figure 5.6: Comparison of the root mean square error of the tardiness in minutes, the travel time duration in hours, and the excess labor time in hours of different solution strategies (without offline solution) on six different instances classes with 20 instances each. We observe that the more sophisticated strategies based on LHP and the route performance decreases the tardiness at the cost of carefully introducing additional travel time (regarding which τ-latest is best) and labor time (where q-earliest is best).

may in practice be acceptable when real costs may be substantially reduced. In Figure 5.6, we visualize the results by boxplots of the tardiness, travel time, and labor time, for the different solution strategies (excluding the offline problem) on the six different instance classes.

We observe that the τ-latest strategy provides the best route performances and therefore smallest travel costs but sometimes runs into troubles regarding tardiness, where a τ-earliest strategy would have been beneficial. Similarly, the q-latest strategy provides the most shift time resources allowing to reduce tardiness, as opposed to the q-earliest strategy. The goal of τ-route is to balance between the extremes of the τ determination strategies considering the load development of the day. Likewise, the q-LHP strategy should provide additional shift resources regarding the flexibility of shift endings times only when necessary. We observe that the τ-route strategy sacrifices a slight amount of route quality in exchange for substantially less tardiness. Likewise, the LHP carefully provides additional labor time to be used to reduce tardiness. Combining both strategies results in a reasonable trade-off over all the instances classes, where a decision maker may also select a suitable combination of strategies given the load and shift structure of the day.

For τ-latest, q-LHP we use $\zeta = 1.2$ after preliminary tuning experiments to convert the route performance values to the average time to serve an order as described in Section 5.7, and for τ-route, q-LHP $\zeta = 1.15$. This is a first effective correction mechanism. Further research is needed regarding the driver performance estimation, especially since the waiting time, the route departure strategy, and the driver performance have an immanent cyclic dependency. The parameters used for the three-parameter inverse power law model to estimate the route performance $\phi(\lambda)$ given the load λ are tuned by means of least squares optimization to $k_l = 23.144, k_m = 3.951, \alpha = 0.174$, with a 25% estimated constant relative standard deviation. For the driver send home strategy, $\tilde{d}$ is set to 55 minutes. The labor time cost factor $\gamma = 4$.

5.10 Conclusions

We considered a same-hour delivery problem variant, a purely dynamic and stochastic vehicle routing problem with short delivery deadlines and shift flexibility arising from a real-world application. Orders arrive at an online store throughout the day, regarding which we have stochastic knowledge by means of hourly estimates. They have to be delivered within only a few hours by drivers deployed at a single depot. The goal is to reduce or evenly spread tardiness if not avoidable among the customers and to minimize travel and labor costs. Drivers may be sent home early or have their shifts extended to some degree to account for the uncertainty of the load.

Our proposed double-horizon approach is able to effectively address the dynamic and stochastic shift planning aspect. The large horizon planning with its simplified order-to-drivers assignment is able to derive meaningful target shift end times. These are exploited in the short horizon planning—the actual routing performed by ALNS—by augmenting its objective function, releasing additional shift resources in an informed way.

Another important aspect is to determine the route departure times, where neither the naive earliest nor the latest strategy suffices. We devised a more balanced strategy that estimates the expected route performance (average time per order in a route) depending on the current load and start routes earlier that are already close to the desired performance and later when the performance is worse.

The combination of both approaches leads to superior performance over the naive strategies or allows for trade-offs regarding substantially reduced travel and labor time versus a slight amount of more tardiness on artificially created instances for different load profiles (business day vs weekend) and shift plans (generous vs tight vs shortage).

Further research is needed to improve the estimation of the driver performance over the day, especially for real delivery areas in a city, also studying the impact of traffic. This is also a basis for the accuracy of the LHP. A difficulty lies in the cyclic dependency between the driver performance estimation and the route departure strategy, where we used a bootstrapping mechanism by fitting a function on idealized randomized instances, incorporating the driver waiting times by a constant factor over the whole day to pragmatically resolve this in a first step.

CHAPTER 6

Route Performance Prediction for Same-Hour Delivery Problems

We are facing again the delivery problem from the previous chapter, a vehicle routing problem where orders arrive dynamically over the day at an online store and have to be delivered within a short time—this time with historical real-world data and not in the real-world inspired simulation. Stochastic information in form of the expected number of orders, the weight of orders, and the traffic congestion level is now available upfront and shall be exploited for an improved planning. More specifically, the goal is to predict the average time needed to deliver an order for a given time and day. This information is desirable for both routing decisions in the short horizon and planning vehicle drivers' shifts with just the right capacity prior to the actual day.

We compare a white box linear regression model and a neural network based black box model on historic route data collected over three months. We employ an hourly data aggregation approach with sampling statistics to estimate the ground truth and features. The weighted mean square error is used as loss function to favor samples with less uncertainty. A mean validation R^2 score over 10×5-fold cross-validations of 0.53 indicates a substantial amount of unexplained variance, likely due to driver bias and higher variance in low-load times of the day. Both predictors are slightly optimistic and produce median standardized absolute residuals of about one.

This work has been presented at the ISM 2020 conference and published in the respective proceedings:

> Nikolaus Frohner, Matthias Horn, and Günther R Raidl. Route duration prediction in a stochastic and dynamic vehicle routing problem with short delivery deadlines. In *Proceedings of the 2nd International Conference on Industry 4.0 and Smart Manufacturing (ISM 2020)*, volume 180 of *Procedia Computer Science*, pages 366–370. Elsevier, 2021

6.1 Introduction

We consider a dynamic vehicle routing problem with stochastic customers arising in a real-world application. Orders are placed at an online store throughout the day and have to be delivered within one or two hours from a single depot. Decisions have to be performed in real-time regarding the clustering of orders, the routes for the vehicles, when to start these routes, and when to send drivers home from their shifts since we have some flexibility concerning shift ends.

Stochastic information per hour regarding the number of arriving orders over the day, their expected weight classes, and the traffic congestion levels is available upfront. The goal is to facilitate this information to avoid myopic decisions for the aforementioned aspects and account for the dynamism of the problem. Additionally, the prior shift planning itself relies on this as we need to have the right amount, possibly including a safety buffer, of driving capacity available over the day.

Our approach is to condense all this stochastic information into one single relation, the mean order delivery time per hour $\phi_t(a)$, which maps for a given day a and hour t to the time needed to serve an order. Conversely, the reciprocal value is the number of orders that can be served per hour.

In this work, we train and evaluate corresponding predictors $\tilde{\phi}_t(a)$ in a supervised learning setting. Different white box and black box models are studied. Error analysis is performed to allow selection between more conservative and more aggressive use of the prediction that naturally comes with uncertainties.

Data on the daily activity has been collected over three months and is used both for training and evaluation in 5-fold cross-validations. Data acquisition was subject to substantial measurement noise due to the manual logging of route legs by the drivers. Both, weighted least squares regression and a neural network (NN) with a weighted sample loss function resulted in the best results on the collected data.

In Section 6.2, we discuss related work on which ours is based. Section 6.3 introduces our linear white box model, whereas Section 6.4 describes the black box model with the neural network. Results are presented in Section 6.5 after which we conclude in Section 6.6.

6.2 Related Work

The dynamic and stochastic vehicle routing problem we consider is presented in detail together with an adaptive large neighborhood search routing algorithm and large horizon planning approach for handling dynamic shift ending flexibilities in Chapter 5. There, also first steps towards predicting order delivery time on simulated data without traffic and stop time variability are performed.

We make use of a classical result by Beardwood et al. [9] from the late 50s which proves asymptotic dependence $\sim k_l\sqrt{n\mathcal{A}}$ of the length of an optimal traveling salesperson

problem tour on the number of cities n, distributed on some geometric area $\mathcal{A}$; k_l is a model parameter depending on the distribution of the cities and the metric.

An extension to the capacitated vehicle routing problem (CVRP) is presented by Daganzo in 1984 [35], accounting for the capacity C of vehicles and the average distance to customers $\bar{r}$ in a delivery area. The suggested model for the length is $2\bar{r}n/C + k_l\sqrt{n\mathcal{A}}$. An intuitive explanation is that n/C routes are needed and that each vehicle has to drive to a dense region of customers yielding round trip distance $2\bar{r}n/C$ and intra-customer distance $k_l\sqrt{n\mathcal{A}}$, following [9].

Figliozzi [51] builds upon these results and proposes further refined white box models to estimate route lengths of classical CVRP benchmarks with and without time windows and a real-world freight forwarding problem. Results with high estimation accuracy are obtained for corner-depot instances by making use of the known number of routes m and adding a term $k_b\sqrt{\mathcal{A}/n}$. The problem is similar to ours with the main difference being that we do not know the number of routes in advance. The same author describes a traffic-aware tour model [52] with optimization constraints that link the number of vehicles needed and the variability induced by congestion to ensure service guarantees with a certain confidence. From the latter, we adopt the approach to model the increase of free-flowing travel times due to congestion by a corresponding factor.

6.3 White Box Model

Let $R_t(a)$ be the set of all routes that start within the hour t on day a. We consider one route $r \in R_t(a)$ with $l_r \geq 1$ served customers $v_1, \ldots, v_{l_r}$. Its total associated duration Δ_r is assumed to consist of the loading time of the goods, the total travel time, and the total stop times at the customers, formally expressed by

$$\Delta_r = \Delta_r^{\text{load}} + \Delta_r^{\text{travel}} + \Delta_r^{\text{stop}}. \tag{6.1}$$

We assume that the mean time per customer Δ_r/l_r is a random variable drawn from an unknown distribution $\mathcal{D}$ with mean $\phi_t(a)$—our ground truth.

As stated before, we seek to create a predictor $\tilde{\phi}_t(a)$. To this end, we propose three features observed in the hour the route is started t and the subsequent hour $t+1$ that have likely the strongest influence[1] on Δ_r: The number of orders due n, the traffic congestion level ξ as the average relative increase of base travel times within the delivery area, and the orders' weight w, each in the given hour.

We assume that the spatial distribution of the orders is stationary over the day. Then, the features n and ξ have supposedly the strongest impact on Δ_r^{travel} but are negligible for the rest. In contrast, w determines the time needed for loading Δ_r^{load} and delivering the goods to the customers at the stops Δ_r^{stop}. Predicting these features are difficult problems in their own right. For this work, we assume corresponding predictors to be given and do not explicitly analyze the uncertainties they introduce.

[1] orders are due within one or two hours after having been placed

Adapting [51, 52] and normalizing the delivery area to $\mathcal{A} = 1$, we propose the following model with parameters to be estimated by (weighted) linear regression:

$$\tilde{\phi}_t^{wb}(a) = k_c + \sum_{t' \in \{t, t+1\}} k_w^{t'} w^{t'} + k_m^{t'} \xi^{t'} + k_l^{t'} \frac{\xi^{t'}}{\sqrt{n^{t'}}} \tag{6.2}$$

The superscript t' indicates the feature with the corresponding parameter for the given hour. We omit this for convenience everywhere else, the use of the features in the current hour t and subsequent hour $t+1$ is henceforth implicit.

Parameter k_c represents the average loading and stop time, independent of any feature. Additionally, k_w accounts for an additional increase in those by the weight or bulkiness of the delivered goods.

Parameter k_m belongs to a traffic proportional term arising from travel times to the first and back from the last customer, loosely corresponding to the term $2\bar{r}/C$ from [35]. We make the simplifying assumption that on average, the maximum number of customers in a route l_r implied by the delivery deadlines is constant for our problem. The actual relation is likely to be more complicated since we expect it to increase with larger n, as more efficient routes can be created and the number of required routes decreases. However, l_r are limited by the finite stop times needed for each customer and the orders' due times implied by the delivery guarantees of one or two hours.

Parameter k_l [9] accounts for the travel times between the customers that are expected to increase with traffic but decrease with the number of orders $\sim 1/\sqrt{n}$, leading to the feature $\xi/\sqrt{n}$.

Significance and contribution to the prediction quality of all the coefficients will be studied in Section 6.5 with linear regression using scikit-learn [115].

6.4 Black Box Model

In contrast to the white box model, we consider a general function with learnable parameters θ

$$\tilde{\phi}_t^{\mathrm{bb}}(a; \theta) = g(w^t, \xi^t, n^t, w^{t+1}, \xi^{t+1}, w^{t+1}; \theta) \tag{6.3}$$

which is realized by a fully connected feed-forward neural network with two hidden layers of 64 neurons each and ReLU activation functions. This network should allow for a piecewise linear approximation of $\phi_t(a)$ with reasonably high resolution which was confirmed by preliminary tests to define its architecture. The traffic congestion factor lies approximately in the range from 1.0 to 1.6, corresponding from free-flowing traffic to heavy congestion with a 60% increase of travel time.

As loss function, the weighted mean squared error is to be minimized, where S^{T} denotes a training set of day-hour tuples

$$L(\theta) \propto \sum_{(a,t)\in S^{\mathrm{T}}} \left(\bar{\phi}_t(a) - \tilde{\phi}_t^{\mathrm{bb}}(a;\theta)\right)^2 / s^2_{\bar{\phi}_t(a)} \tag{6.4}$$

and $\bar{\phi}_t(a)$ denotes the sample mean by averaging over the Δ_r/l_r occurring in (a,t). According to the central limit theorem, the sample standard error $s_{\bar{\phi}_t(a)}$ decreases with the inverse square root of the number of routes. Since $\phi_t(a)$ itself is unknown, hours, where less variance and more routes are observed are given more weight in the loss function than those with higher uncertainty.

We implemented the neural network approach using TensorFlow [1] with the Adam [90] optimizer. Early stopping monitoring the performance on 30% validation data is used for regularization. As initial learning rate we choose 0.01 with dynamic updates by factors of one-tenth down to 10^{-4} when plateaus are encountered.

6.5 Results

We have prepared data by aggregating routes for day-hour tuples collected over three months. Routing is performed in a semi-automatized way by experienced dispatchers with the help of a routing software. Travel times to the customers and stop times at the customers are logged by the drivers, whereas the loading time and the travel time back to the depot are estimated.

The features vector is given as $(w,\xi,n)_{(a,t)}$, where w is estimated by calculating the hourly average for the same weekdays (or holidays as own type) over the month, ξ is determined by comparing the base driving times as provided by the HERE[2] public routing API with the logged driving times of the routes and taking the average for each (a,t), and n is taken to be the number of orders due in (a,t). Since routes started at the end of an hour are likely to be more influenced by the subsequent hour, we always present $(w,\xi,n)_{(a,t+1)}$ to our models as well.

Routes with implausible logs were discarded, and a quantile cut of $[0.02, 0.98]$ on the congestion level ξ is applied to remove routes with unrealistic driving time logs and outliers. This reduces the number of routes used for aggregation down to 87%.[3]

The labels are calculated as

$$\bar{\phi}_t(a) = \frac{1}{|R_t(a)|} \sum_{r\in R_t(a)} \Delta_r/l_r. \tag{6.5}$$

The sample standard error $s_{\bar{\phi}_t(a)} = s_{\phi_t(a)}/\sqrt{|R_t(a)|}$ is used for weighting the labels as done in Eq. (6.4).

[2] https://developer.here.com/documentation/routing-api/dev_guide/index.html
[3] the total number of routes is not revealed

Table 6.1: Standardized residuals' R^2 scores, medians (Med), and median absolute deviations from the medians (MAD) averaged over ten 5-fold cross-validations with different models and feature ablation.

model	$\overline{R^2_{\text{train}}}$	$\overline{R^2_{\text{val}}}$	$\overline{\text{Med}^{\text{res}}_{\text{train}}}$	$\overline{\text{Med}^{\text{res}}_{\text{val}}}$	$\overline{\text{MAD}_{\text{train}}}$	$\overline{\text{MAD}_{\text{val}}}$
WLS-$\xi/\sqrt{n}$,ξ,w	0.52	0.50	-0.34	-0.34	**1.37**	1.39
WLS-$\xi/\sqrt{n}$,ξ	0.52	0.50	-0.34	-0.34	**1.37**	1.39
WLS-$\xi/\sqrt{n}$, w	0.47	0.45	-0.43	-0.43	1.44	1.45
WLS-$\xi/\sqrt{n}$	0.47	0.45	-0.43	-0.43	1.44	1.44
WLS-$1/\sqrt{n}$	0.36	0.34	-0.52	-0.52	1.51	1.52
WLS-ξ	0.28	0.27	-0.63	-0.63	1.46	1.46
NN-n,ξ,w	**0.53**	**0.53**	**-0.32**	**-0.32**	1.39	**1.38**
NN-n,ξ	0.52	0.51	-0.33	-0.33	1.41	1.39
NN-n	0.40	0.39	-0.47	-0.46	1.51	1.50

As first sanity checks, Pearson correlation coefficients are calculated yielding moderate correlations $\rho_{1/\sqrt{n},\bar{\phi}_t(a)} \approx 0.6$ and $\rho_{\xi,\bar{\phi}_t(a)} \approx 0.5$. Correlation between the weight w and $\bar{\phi}_t(a)$ is much smaller, therefore we further check for significance in a t-test of a linear regression $\left(\overline{\Delta^{\text{stop}}/l}\right)_t(a) \sim k_c + k_w w + k_m \xi$. We include also the traffic to check how it contributes to the explanation of the stop times per customer when competing with the weight. For k_w this reveals significance with a t-score of almost 10 and no significance for k_m with a t-score only slightly above zero. We are in the range where the t-score approximately equals the z-score.

Table 6.1 compares average training and validation performance values on 10×5-fold cross-validations (hence $N = 50$). Weighted least squares (WLS), the white box model, is compared with the neural network based black box model. Only pairs (a, t) with at least four routes are considered to have somewhat valid sample statistics, leading to the total batch size of 1081. We measure weighted R^2 scores, median residuals to measure prediction bias, and median absolute deviation of the median (MAD). We make use of robust statistics since our problem is subject to outliers. As an ablation study, we remove each feature individually and observe that for WLS the order weights w have no impact on our figures. This is backed up by the fact that t-tests on the significance of coefficients in the linear model yield scores close to zero for k_w when combined with and seemingly dominated by the other features. In contrast for the NN, we see a slight increase in the performance values when going from n, ξ to n, ξ, w.

As expected, using either n or ξ without the other feature hurts the performance substantially. The neural network achieved the best performance averaged over the 10×5-fold cross-validation runs with an R^2 score of 0.53, median of -0.32, and median deviation from the median of 1.38 on the standardized residuals.

By comparing training and validation figures, we observe very little, if any, overfitting. One reason for this is conjectured to be the substantial amount of noise in the data which makes overfitting with the given small amount of features and small (WLS) to moderate (NN) model capacity quite difficult. Possible noise sources are manifold and include the

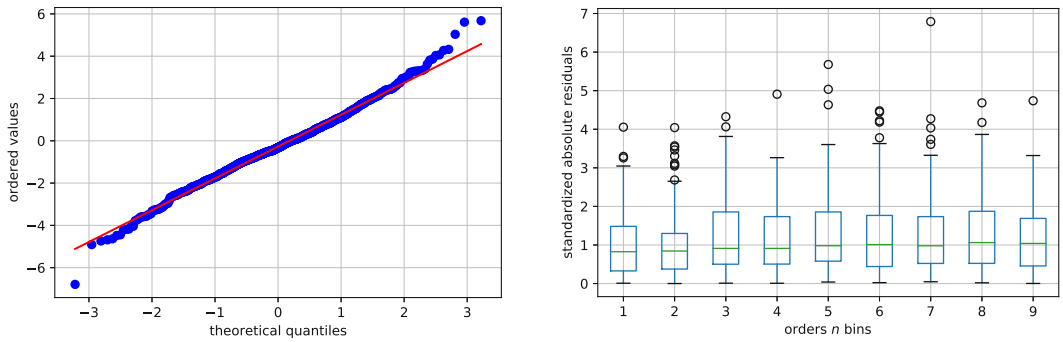


Figure 6.1: Left: QQ-plot for standardized residuals of neural network prediction on whole data batch. Right: Standardized absolute residuals of neural network prediction on whole data batch over bins of number of orders n.

driver bias (each driver has a different pace), non-stationarity of the spatial distribution of the orders, traffic variability over the delivery area, manual leg duration measurement error, and the hourly estimation of the features.

The models consistently underestimate the time needed to serve an order, as indicated by the negative bias. Figure 6.1 left depicts a QQ-plot of the standardized residuals (using the sample standard errors) vs. a normal distribution. We observe a good resemblance with a normal distribution (linear fit with $R^2 = 0.996$) with the mentioned slight negative bias. The standardized absolute residuals over the number of orders n bins are depicted in Figure 6.1, right. We observe quite stable performance over n with median standardized absolute residuals around one.

In conclusion, both model types with all features have comparable performance, whereas the NN model shows marginally but consistently better validation performance. This is illustrated by a comparison of both predictors in action on example days in Fig. 6.2. Some outliers regarding the mean order delivery time are observed on this day, smoothed out by the predictors. The behavior is not exactly the same, yet they are quite similar. This leads us to the conclusion that data and feature bias are dominating and both models act approximately equivalent in our high-noise setting on our historical data. Still, the neural network does not show reasonable (asymptotic) behavior for unseen regions of the features, in particular n, which is the advantage of the white box approach.

6.6 Conclusions

We analyzed the performance of a linear white box model and a neural network based black box model on a route duration prediction problem in a stochastic and dynamic vehicle routing problem. It is derived from a real-world setting where orders arrive in an online store dynamically over the day and have to be delivered to customers within short due times. The goal is to predict the mean order delivery time for a given day and hour. This information is required for improved routing decisions and planning the number of required drivers over the day. Stochastic information per hour is given regarding the number of arriving orders, their average weight, and the traffic congestion level.

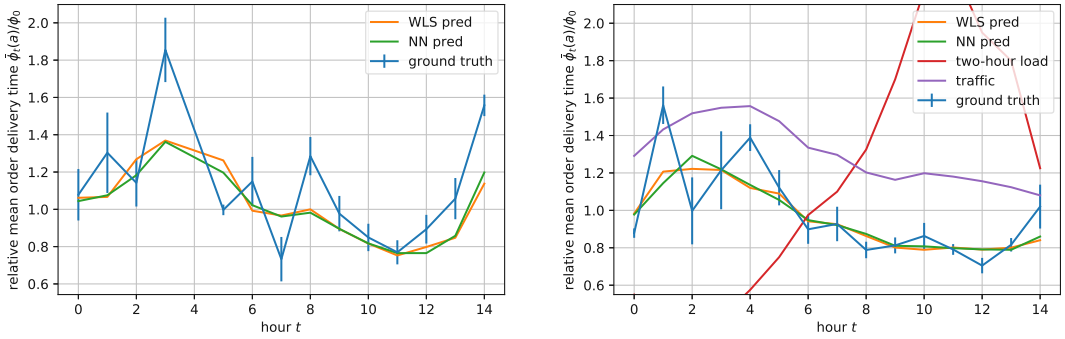


Figure 6.2: Example days, comparing the ground truth in form of the relative mean order delivery time $\bar{\phi}_t(a)/\phi_0$, with ϕ_0 a total average, with the NN and the WLS predictions. Right including the relative two-hour load and relative traffic to see their influence on the driver performance.

Data has been collected over three months by manual logging of travel and stop times by the drivers, subject to substantial measurement noise. The ground truth and the traffic feature were accessed by sampling statistics and subject to noise themselves, depending on the number of routes in a given hour. Hence we chose the mean squared error weighted with the inverse sample standard errors. Based on results from the literature, we proposed a linear model with parameters to be tuned by weighted linear regression and a small two-layer neural network.

In an ablation study, we showed the importance of combining both the number of orders and traffic congestion level as features. All features were presented to the models for the current and the subsequent hours. Order weights have significant impact on the stop times at the customers, but did contribute little to the overall predictive performance.

In the end, the neural network achieved the best performance with the cost of unreasonable predictions for regions where no or not much training data was available. Smoother approximation with reasonable asymptotic behavior is achieved with the weighted linear regression approach, which also gives a comparable performance. Standardized absolute residuals are both stable with a median about one over increasing numbers of orders.

The real-world performance of the predictors is an open question. High-quality data collection including traffic information from a third-party source would be recommendable to resolve travel time and stop time effects. The number of orders that are visible when a route is started should be logged to learn how the number of orders feature could be improved. One possible criticism of the suggested weighted loss functions is that hours with more routes are likely hours with better performance, which is conjectured to result in the negative (optimistic) bias of the predictors. Naturally, the less busy hours are harder to predict, since less data is available and the variance is intrinsically higher. This can be mitigated by employing a safety buffer based on the error analysis of the predictors.

Further work is concerned with refining the models on simulated data where noise sources can be excluded and further validation on newly collected data.

CHAPTER 7

Learning Surrogate Functions for the Short-Horizon Planning in Same-Hour Delivery Problems

As we have learned so far about same-hour delivery, timely routing decisions have to be made over the planning horizon of a day. The well-known sampling approach from the literature for considering expected future orders is not suitable due to its high runtimes. To mitigate this, we suggest using a surrogate function for route durations that predicts the future delivery duration of the orders belonging to a route at its planned starting time. This surrogate function is directly used in the online optimization replacing the myopic current route duration. The function is trained offline by data obtained from running full-day simulations, sampling and solving a number of scenarios for each route at each decision point in time. We consider three different models for the surrogate function and compare them with a sampling approach on challenging real-world inspired artificial instances. Results indicate that the new approach can outperform the sampling approach by orders of magnitude regarding runtime while substantially reducing travel costs in most cases.

This work has been presented at the CPAIOR 2021 and published in its proceedings:

> Adrian Bracher, Nikolaus Frohner, and Günther R Raidl. Learning surrogate functions for the short-horizon planning in same-day delivery problems. In *17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR'21)*, volume 12735 of *LNCS*, pages 283–298. Springer, 2021

It has also been published and extended further to an iterative learning approach in Adrian Bracher's bachelor thesis [22].

7.1 Introduction

Short delivery times are essential when it comes to selling goods online, especially during the COVID-19 pandemic when many physical stores had to close temporarily. An increasing number of online retailers are offering same-day delivery to satisfy the demand for quickly available goods, further intensifying the need for cost and labor-efficient dynamic vehicle routing. Same-day delivery problems [162] are stochastic and dynamic in nature and are a subcategory of vehicle routing problems. We again consider the problem variant from Chapter 5: Orders arrive dynamically over the day and are due only a short time after arrival. The orders have to be assigned to drivers and routes are generated with the goal to minimize delivery tardiness, travel times, and labor costs of the drivers involved. The fleet is homogeneous and the orders are served from a single depot. Each driver has a predefined shift, however, the shift end times can be advanced or postponed to some extent to account for the uncertainty of the actual load.

The presented double-horizon approach focused on the dynamic shift planning aspect. As its use in practice has shown, the existing short-horizon optimization may sometimes lead to unsatisfactory results, which we declare myopic, due to the following aspect: Routes that are optimal regarding all available orders sometimes have to start soon due to some orders, but also include one or more less urgent orders with a delivery deadline relatively far in the future. If these currently available orders with later deadlines introduce a significant travel overhead, it would frequently be wiser to postpone their delivery as they can likely be combined with future orders resulting in more efficient routes overall. Thus, it would be beneficial to split routes between urgent and less urgent orders. Routes can only be changed up to their departure, and possible future improvements for routes with a later starting time are not considered in the static, myopic optimization. The aim of this work is to present our adaptations to improve on this aspect.

The basic idea of our approach is to craft a function that discounts travel times based on the aforementioned observations, making separate routes with later starting times more attractive. We will refer to that function as a surrogate, since it replaces the normal route duration in the objective function and also is used instead of a classical sampling approach, which is the de facto standard for stochastic considerations. This surrogate function is trained offline in a supervised learning fashion, reducing the computational effort in the online application in comparison to a sampling approach substantially. The necessary training data is generated by full-day simulations, in which we sample and solve 100 scenarios for each route at every decision point in time. Three different models for the surrogate are considered, a manually crafted exponential function, a linear regression, and a multi-layer perceptron.

In Section 7.2, an overview of related work is given and discussed. Section 5.3 of the chapter introducing the same-hour delivery problem already defines and formalizes the problem at hand. We illustrate the problem of myopic short-horizon routing in Section 7.3. Then, in Section 7.4, we explain our new approach in detail and describe the training data acquisition and training process in a step-by-step manner. Details on our test setup,

and a comparison of our approaches with a sampling approach on real-world inspired artificial instances, can be found in Section 7.5. We observe that on our benchmark instances, the new approach reduces route travel costs by up to ≈8% in the median compared to the myopic optimization with similar tardiness. The sampling approach, in comparison, achieves a similar route duration reduction but requires a computation time that is larger by a factor of ≈540. We finally conclude in Section 7.6.

7.2 Related Work

For a review on dynamic and stochastic vehicle routing problems, see, e.g., Ritzinger et al. [126] and also the related work section 5.2. Our underlying problem variant is introduced in [55], see also Section 5, and derived from an online store with promised delivery durations of one or two hours. The problem is fully dynamic with a planning horizon of one day, where stochastic information regarding the hourly number and spatial distribution of orders is available upfront. A distinguishing feature is a certain flexibility of the shift ending times of the drivers, which is considered in the objective function together with route durations and a penalty for delivery deadline violations.

So far, the pillars of solving this problem are an adaptive large neighborhood search (ALNS) [127, 117, 8] for the repeated point-in-time optimization runs to obtain routes for currently available orders and a dual horizon approach inspired by Mitrović-Minić et al. [109]. At every decision epoch, a simplified assignment problem is solved for the larger horizon (i.e., the whole day) using expected values for the orders and driver performance. This allows an estimation of the required labor time which is subsequently fed back into the objective function used in the point-in-time optimization runs considering only short horizons. Near real-time decisions regarding planned assignments of orders to multiple trips of drivers, when to send drivers home, and when to start routes are then derived from the result of this optimization.

Due to the short delivery deadlines within a few hours after customers place their orders, the problem falls into the class of *same-day delivery problems* (SDDP). In a recent notable work, Voccia et al. [162] present an SDDP variant with hard time windows where orders can also be delegated to a third party, apart from delivering it with the in-house fleet. The number of orders to be delivered in-house is to be maximized and formulated as reward of a Markov decision process (MDP). A multiple scenario sampling approach (MSA) [11] is tailored to the problem, where at every decision epoch a multi-trip team orienteering problem with time windows is solved heuristically and a consensus solution is derived. This method increases the number of filled requests for some instance classes substantially compared to a simple delay strategy, where decisions are postponed to gather more information until an impact on the number of filled requests occurs. Still, a relevant drawback is that at every decision a couple of minutes computation time is required, making it unsuitable for our near real-time setting.

Although we do not model our problem as an MDP explicitly, we perform implicit state transitions where actions (for each driver in the depot either wait, start an unalterable

delivery route, or end shift) are derived from the heuristic solution. The goal of this work is to further adapt the objective function so that the implied actions lead to states with a higher expected reward in the near future.

Using the approximate dynamic programming paradigm [119], Ulmer et al. [150] solve a single-vehicle SDDP with preemptive depot returns using an Approximate Value Iteration (AVI) scheme where the value function is learned in an offline training phase over full-day simulations. Furthermore, a dynamic state space aggregation is used to create a lookup table facilitating near real-time online decision making.

Joe and Lau [89] build upon this approach for a dynamic vehicle routing problem with stochastic customers and different degrees of dynamism where re-routing decisions have to be performed on routing plans over the day. They replace the lookup table with a deep Q network employing value function approximation via temporal difference learning with experience replay. A heuristic search in the decision space is performed via simulated annealing. This approach is compared with AVI [150] and MSA [11], and the authors report reductions of the costs in the range of 10% for higher degrees of dynamism.

In a similar spirit, we approximate the value of states by predicting the future costs of orders that are in a currently planned route utilizing parametric functions to be learned in an offline training phase with training data derived from multiple realizations in the short horizon—we consider the routes separately and do not roll-out until the end of the day, hence we make use of a vehicle-based and temporal decomposition. This learned function is then incorporated as a surrogate in the objective function to be solved heuristically using our ALNS (Section 5.4), resulting in more anticipatory online decision making.

Another recent work related to highly dynamic delivery is the restaurant meal delivery problem discussed by Ulmer et al. [151], a pickup and delivery problem with stochastic customers and random ready times. They make use of *cost function approximation* [120, 109] to implicitly consider the future impact of decisions and ready time uncertainties by augmenting the assignment-guiding objective function with a time buffer and a postponement strategy. The time buffer is tuned by offline simulations and subsequently used in an online comparison of different policies.

As a similar idea in the realm of dynamic combinatorial optimization problems, Dickerson et al. [40] propose learnable potentials for structural elements of graphs in a dynamic matching problem applied to kidney exchange to capture their future usefulness. The potentials are then incorporated in the initially myopic objective function for the point-in-time optimization, leading to more matches overall.

Our approach also relates to surrogate-based optimization [97] applied to engineering problems for which the objective value $f(\boldsymbol{x})$ of solutions in a continuous design space is potentially noisy and expensive to evaluate, and therefore approximated by a faster surrogate model $\tilde{f}(\boldsymbol{x})$.

7.3 Illustrative Example

To make the issue we address in this work clear, we present a simple example of a DYN-DAY instance, in which an optimal solution to a first DYN-$\tilde{t}$ instance leads to a situation so that an overall suboptimal solution for DYN-DAY is achieved.

Let us assume orders 1–6 become available at $\tilde{t}$=0 and we are thus considering DYN-0. Orders 1–5 are supposed to have the same due time 60 minutes later. The remaining order 6 is located far away from orders 1 to 5 and has a substantially later due time of 120 minutes. Considering only these orders an optimal solution to DYN-0 would be to pack all orders into one route since only then the total route duration is minimal. This single route r_1 has to start at time $\tau_{r_1} = 5$ to avoid any tardiness. This solution is depicted in the top half of Fig. 7.1. The problem with this solution arises when half an hour later new orders 7–10 are placed with some delivery locations close to order 6, which itself, however, has been included in the already started first route. An optimal solution for DYN-30min will then be a route r_2 with the remaining orders 7–10 as also pictured in Fig. 7.1. The resulting total objective value, which in this case is equal to the sum of all route durations, is 220 minutes.

A better solution to this example can be seen in the second half of Figure 7.1. The important difference is that the first route from the previous solution is split into two, resulting in one route r_3 comprising orders 1–5, starting at time $\tau_{r_3} = 5$ and having a mean order duration better than the former single route, and one route r_4 containing only order 6. This latter route has a bad mean order duration of 50 minutes per order, but also a much later starting time of $\tau_{r_4} = 90$. Even though this results in a worse short-term objective value for DYN-0, this second route has now a lot of slack left and considering expected future orders can likely be improved later. In our example, this happens when the new orders 7–10 become available in DYN-30, and order 6 can be delivered in one route r_5 together with the new orders. Overall the objective value and sum of all route durations for this solution is 190, which is an improvement of approximately 14% over the myopic solution.

In conclusion, when orders are expected in the near future, it makes sense to postpone to a certain degree the delivery of orders with due times farther in the future when they cannot be well integrated into soon-to-start routes.

7.4 Discounting Travel Times to Consider Expected Orders

As pointed out in Section 7.2, to at least partially avoid traps like the one sketched above arising from the myopic view of the DYN-$\tilde{t}$ instances, the standard method from literature is to sample scenarios into the near future by creating artificial orders from expected spatial and temporal distributions, to solve these scenarios, and then to derive a consensus solution [11, 162]. We propose the simpler approach of discounting durations of routes in the objective function of the DYN-$\tilde{t}$ instances in dependence on their starting

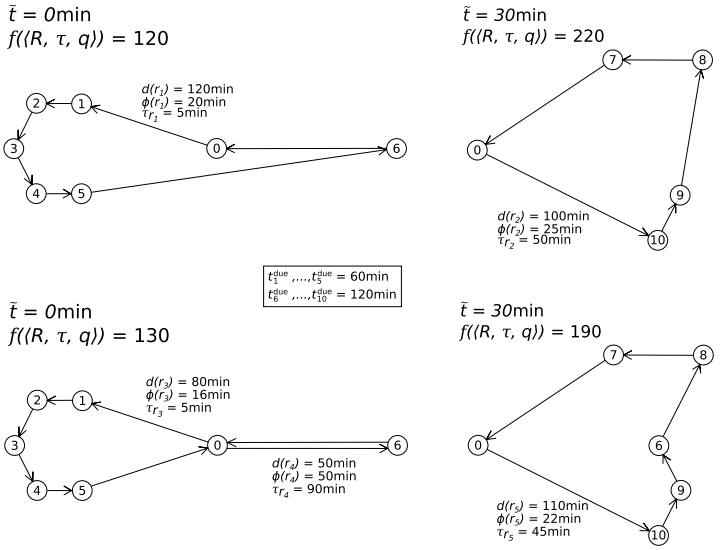


Figure 7.1: Myopic solution (top) vs. optimal solution (bottom). Node 0 represents the warehouse, all other nodes orders.

times, the number of expected future orders, and further features. We make use of supervised learning to come up with a surrogate function for the route durations to move the computational effort into a one-time offline training phase. This function is then directly used in the optimization of a given DYN-$\tilde{t}$. We now describe our approach in detail.

In the definition of $f(\langle R, \tau, q\rangle)$ in (5.4) on page 109 we replace the route duration $d(r)$ of each route $r \in R_u,\ u \in U$ with a *discounted duration* $d'(r)$ acting as a surrogate for the future delivery time of the orders belonging to r. We define the following aims to guide us to a sensible discounting function:

- Routes that are already efficient, i.e., have a low mean order duration $\phi(r)$, should not be modified.
- The discounting of the duration should in general be stronger when more orders are expected in the near future. On the contrary, we should not reduce $d(r)$ if there are no further orders expected until route r should start.
- Routes that are inefficient and combine orders that are due soon with orders that have significantly more time left should be avoided in particular.
- In conclusion, the discounted route duration $d'(r)$ should approximate the *expected total time it will take to perform the deliveries of that route in the future*, taking into account expected new orders and assuming optimal routing decisions also in the future.

A current route that will be started soon cannot be expected to be improved much as not many new orders are expected. This includes routes with small slack $\max(0, \tau^{\max}(r) - \tilde{t})$ but also any other case in which the route is started soon, e.g., due to an earliest starting time strategy. In contrast, larger improvements are likely for any route that is planned to be started much later and which is not yet efficient, particularly if many orders are expected in the near future, more precisely in the time interval from the current time $\tilde{t}$ to the route's planned starting time τ_r. Thus, this duration is an important parameter of the discounting.

Another important parameter is the expected number of arriving new orders until the start of the route, i.e., $\omega(\tilde{t}, \tau_r)$. Moreover, the estimated mean order duration of a good DYN-DAY solution $\hat{\phi}$ is also important for the following consideration. A route r to be started at some distant time τ_r and whose mean order duration $\phi(r)$ is worse than $\hat{\phi}$ can be expected to be adapted and combined with future orders so that the average times for delivering the orders in r approaches $\hat{\phi}$.

Formally, we model this by the *discounted route duration function*

$$d'(r) = \begin{cases} g_\Theta(d(r), l_r, \omega(\tilde{t}, \tau_r), \hat{\phi}, \ldots) & \text{if } \tau_r > \tilde{t} \wedge \phi(r) > \hat{\phi} \\ d(r) & \text{else.} \end{cases} \tag{7.1}$$

where function g_Θ represents a machine learning model with trainable parameters Θ and input features that include at least $d(r)$, $l_r, \omega(\tilde{t}, \tau_r)$ and $\hat{\phi}$. This model is supposed to yield reduced durations within $[\hat{\phi} \cdot l_r, d(r)]$ for routes that are not started immediately ($\tau_r > \tilde{t}$) and where the current mean order duration $\phi(r)$ is worse than $\hat{\phi}$. In Section 7.4.2 we will consider three different approaches for realizing g_Θ, which are an exponential function, a linear regression, and a multilayer perceptron.

An aspect of this approximation that deserves mentioning is that multiple routes of the current solution may be scheduled at overlapping times in the future and may compete for new orders. This may slow down the improvement of inefficient routes but may also create new possibilities for more efficient combinations. As we do not see any meaningful and efficient way to consider this aspect and also conjecture that the benefits and disadvantages of concurrent routes in conjunction with the route improvement potential may outweigh each other at least to a certain degree, we do not explore this further here. Moreover, the actual impact may be partially mitigated by suitably tuning Θ.

7.4.1 Obtaining Training Data

To obtain training data for our route duration discounting models, we apply the following sampling-based approach on a set of representative historic or artificial DYN-DAY training instances.

1. We consider a DYN-DAY instance and iteratively solve the implied DYN-$\tilde{t}$ instances in the classical way without any route distance discounting. For each obtained

DYN-$\tilde{t}$ solution, we apply a decomposition approach, in which we consider each route independently by the following steps.

2. Each route r to be started not immediately, i.e., at some time $\tau_r > \tilde{t}$, and for which $\phi(r) > \hat{\phi}$, we create n_{sample} scenarios, with n_{sample} being a strategy parameter. Each scenario consists of the original orders of route r and $n_{\text{orders}} \sim \mathcal{P}(\omega(\tilde{t}, \tau_r))$ additional artificial orders, where n_{orders} is a random number always sampled anew from the Poisson distribution $\mathcal{P}(\omega(\tilde{t}, \tau_r))$ with mean $\omega(\tilde{t}, \tau_r)$. The motivation here is that the arrival of orders can be seen as a Poisson process. Each artificial order is assigned a randomly sampled geographical location from a set of sufficient size representing the delivery area, a random availability time in $(\tilde{t}, \tau_r]$, and a due time that corresponds to the availability time plus the maximum delivery duration promised to the customers. Each scenario created this way is then solved as an independent OFF instance.

3. In each obtained scenario solution we consider each original (i.e., not sampled) order and take its route's mean order duration. The sum of these times over all original orders is then said to be the scenario's total delivery duration for the original orders of route r. Ultimately, we average these total delivery durations over all scenarios to obtain the target duration $\hat{d}(r)$ which we want to approximate by our discounted route duration $d'(r)$.

4. We store the original route r together with $d(r)$, $\tilde{t}$, τ_r, $\omega(\tilde{t}, \tau_r)$, $\hat{\phi}$, and the obtained $\hat{d}(r)$ for training and continue by processing all further routes in the same way.

7.4.2 Models for the Discounting

As introduced in Eq. (7.1), function $g_\Theta(d(r), l_r, \omega(\tilde{t}, \tau_r), \hat{\phi}, \ldots)$ is a trainable model that yields the discounted route duration when $d(r) > \hat{\phi} \cdot l_r$. For training this model we apply the mean squared error (MSE) with respect to the training targets, i.e., $\hat{d}(r)$, as loss function. We investigate here three alternative models presented in the following.

Exponential Function.

$$g_\rho^{\exp}(d(r), l_r, \omega(\tilde{t}, \tau_r), \hat{\phi}) = d(r) - (d(r) - \hat{\phi} \cdot l_r) \cdot (1 - e^{-\rho \cdot \omega(\tilde{t}, \tau_r)}) \tag{7.2}$$

This function was manually crafted based on the previously explained considerations that the mean order delivery time of orders in a current route with a distant starting time can be expected to improve up to a certain amount. The expected maximum improvement is assumed to be equal to $d(r) - \hat{\phi} \cdot l_r$. However, actual improvement can only occur with additional orders in the interval $(\tilde{t}, \tau_r]$. This is expressed by the last term in the function, where parameter ρ controls the speed of approaching $\hat{\phi} \cdot l_r$ in dependence of the number of expected upcoming new orders $\omega(\tilde{t}, \tau_r)$ until the route's starting time τ_r in an exponential manner. The parameter that needs to be learned here is just $\Theta = \rho$, and we apply grid search to find a value minimizing the MSE.

Linear Regression.

Our second approach is a linear combination of a larger set of manually selected features, i.e., linear regression, with the trainable parameters vector Θ being the respective regression coefficients. We initially consider the following features in addition to a constant bias.

1. The basic features $d(r)$, l_r, $\omega(\tilde{t}, \tau_r)$, and $\hat{\phi}$ as in the exponential function.
2. The relative starting time of the route $\tau_r - \tilde{t}$.
3. The difference $\phi(r) - \hat{\phi}$, i.e., how far off the route's mean delivery duration is from the assumed target value $\hat{\phi}$.
4. The *variance of the geographic locations of the orders for each route*, denoted by $\text{var}(r)$; the farther apart the delivery locations are, the more likely it seems that a new order fits nicely in between two existing orders.
5. The square and the logarithm of each of the above features to also accommodate nonlinear dependencies in a simple form.

To avoid the inclusion of features that do not significantly improve the prediction and reduce the danger of overfitting, we started off with just the basic features and iteratively added a feature from the remaining pool that reduced the MSE the most. This process of selecting features was continued until the MSE did not change by more than one percent. 5-fold cross validation was used in this feature selection process to reduce the risk of overfitting. Ultimately, we came up with the feature vector $(d(r), l_r, \omega(\tilde{t}, \tau_r), \log(\omega(\tilde{t}, \tau_r)), \hat{\phi}, \phi(r) - \hat{\phi}, (\phi(r) - \hat{\phi})^2, (\tau_r - \tilde{t})^2, \log(\tau_r - \tilde{t}), \text{var}(r), \text{var}(r)^2)$ used in all further experiments.

Multilayer Perceptron.

Our third model for discounting travel durations is a multilayer perceptron (MLP). It is fully connected with two hidden layers and ReLU activation functions in all layers, and Adam [90] is used as optimizer. The considered pool of features was the same as in the linear regression, and the same selection process was performed leading to the feature vector $(d(r), \log(d(r)), l_r, \hat{\phi}, \tau_r - \tilde{t}, \phi(r) - \hat{\phi})$ used in all following experiments. Note in particular that here the variance of the orders' geographic locations did not show a significant contribution and therefore was not included. Further details on the network configuration and training will be provided below in the experimental results.

7.5 Computational Study

All algorithms were implemented in Python 3.8. Training and evaluation of the regressors was performed with `scikit-learn` version 0.23.1. All tests were conducted on Intel Xeon E5-2640 2.40 GHz processors in single-threaded mode and a memory limit of 4 GB.

In all tests, a driver is sent home as early as possible, i.e., after the driver's last so far planned route or at the earliest shift end, to minimize labor costs. Planned routes always start at the latest possible departure time which does not increase the costs for labor time and tardiness to utilize the full slack for possible improvement. The three different discounting models are compared with results using the myopic optimization as done in [55] and the sampling approach with consensus function. The ALNS, which is the fundamental optimization method for all mentioned approaches, stops after 100 non-improving iterations, and we refer to [55] for all further details concerning its operators and configuration.

7.5.1 Instances

We consider artificial DYN-DAY instances that are inspired by real-world instances of an online retailer. We consider steady, linearly rising, and falling load patterns over 11 hours, where the average load over the day is either 10, 20, 30, or 40 arriving orders per hour. Orders are due in one hour with 60% probability and with 40% in two hours. The order locations are uniformly distributed in the unit square. Travel times between orders are determined by the Euclidean distance multiplied by 50 minutes, additional constant six minutes stop times at the customers, and small loading and postprocessing times from and to the warehouse. The warehouse location is randomly chosen from $\{0.25, 0.75\}^2$ inspired by the slight off-center location of the real-world situation. Since we focus on the route duration costs, sufficiently many drivers are available all the time to ensure zero or very little tardiness. We generated 240 instances in total, 20 for each of the 12 instance classes and perform a 50/50 training and test split. The mean performance $\hat{\phi}$ is provided for each class and calculated by performing full-day simulations with the myopic ALNS. All instances were made available on GitHub.[1]

7.5.2 Training of the Discounted Route Duration Models

Following the training and test data generation as described in Section 7.4.1 applied on the 120 full-day training instances, we end up with a 60% batch of 33 790 training samples and a 40% batch of 22 527 test samples[2] to train and evaluate an estimator for $\hat{d}(r)$.

We train the learnable parameter ρ in the exponential model (7.2) by means of a grid search. The result can be seen in Figure 7.2, which displays how the MSE changes depending on ρ. Moreover, the instance's $\hat{\phi}$ is reduced by 20%, which was empirically determined to produce better results in previous experiments. The motivation behind this is that the $\hat{\phi}$ were derived from full-day runs with myopic ALNS and we actually aim to improve this performance, a desired lower $\hat{\phi}$. The single global optimum for ρ is 0.091, at which the test MSE is 154 909 and 154 265 for the training batch.

[1] https://github.com/nfrohner/pdsvrpddsf

[2] not to be confused with the test set for the full-day simulations

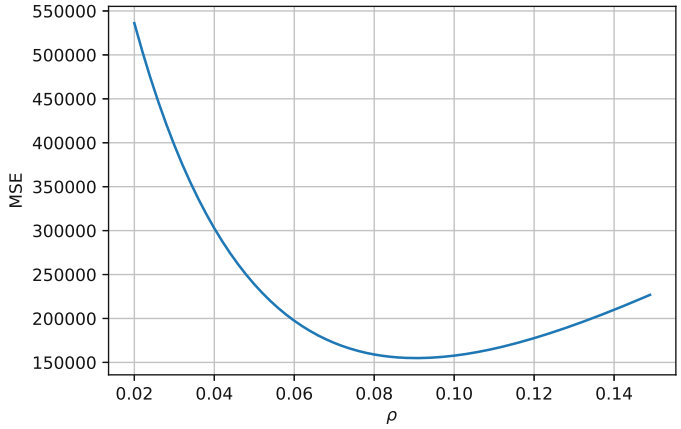


Figure 7.2: Exponential model: MSE of predicted values $g_\Theta(r)$ with respect to labels $\hat{d}(r)$, i.e., the loss over ρ.

In the case of the linear regression with the finally selected features as laid out in Section 7.4.2, MSEs of 144 785 and 143 329 were achieved on the training and test portions of the data, respectively.

Concerning the MLP, preliminary tests suggested that two hidden layers with 50 nodes each seem to be a reasonable choice, which we used further on. The learning rate that is used for training is a constant 0.001. To avoid overfitting we utilize early stopping, for which 300 iterations without improvement of a 10% validation set is the stopping criterion. The resulting training and test MSEs are 80 032 and 79 219 respectively, slightly less than half of the error of the exponential model.

Concerning the MSEs, we can conclude that the linear regression performs slightly better than the exponential model, but the MLP is clearly superior. As we considered separate training and test sets and the respective MSEs lie close together for all three models, we conclude that overfitting seems to be no issue for all three models.

7.5.3 Full-day Simulation Results

The myopic short-horizon optimization serves as a baseline to quantify the improvement that is achieved. Furthermore, the three route duration discounting approaches are compared to a sampling approach with consensus function as the de facto standard for considering stochastic aspects. This approach creates for each DYN-$\tilde{t}$ instance 100 scenarios by augmenting the original instance with randomly sampled orders. These sampled orders are generated in the same manner as already explained in Section 7.4.1, except that the time interval $[\tilde{t}, \tilde{t}+1h]$ is used instead of the slack of the route, i.e., samples are generated for up to one hour into the future. These scenarios are then solved with the myopic short-term optimization utilizing ALNS. Then, all sampled orders are removed from each scenario solution. Finally, a consensus solution is derived from the scenario solutions in a way that was inspired by [162]. The selection is done by counting identical scenario solutions and choosing the most frequent solution as a consensus solution. We

define identical in this context as two solutions that assign identical routes to the same drivers in the same sequence. Analogous to that identical routes are defined as routes that contain the same orders in the same sequence.

We use the original objective function $f(\langle R, \tau, q\rangle)$ as defined in (5.4) as the primary measure of success for comparing results, but also aim to gain a more in-depth understanding of the different approaches by observing the total duration of all routes in a solution, the total excess labor time of a solution, the mean order duration over the whole solution $\bar{\phi}$ and the running time on the specified test setup. A tardiness penalty factor of 1 000 is chosen to approximate a lexicographic approach by a weighted sum. The excess labor times cost factor γ is set to 4. Tardiness is not presented in this Section, because it is negligibly small for all instance classes and approaches alike, which was one of our aims when generating the test instances as explained in Section 7.5.1.

In Table 7.1 the median of the mentioned measures of success are compared for all instance classes and the median of the relative changes to results of the myopic approach is displayed for the most important measures as well. As the sampling approach did not terminate within a time limit of 700 hours per full-day instance for average loads of 30 and higher, we only obtained results up to an average load of 20 for it. In Figure 7.3 boxplots of $f(\langle R, \tau, q\rangle)$ are drawn over instance classes grouped by the average load as well as the load pattern.

As expected, all approaches that consider possible future orders outperform the myopic optimization, up to 8% in the median. We observe that the exponential approach outperformed the other approaches for average loads of 30 and 40. Furthermore, a positive correlation between the average load and the relative improvement over the myopic short-term optimization can be seen. Falling load solutions have higher $f(\langle R, \tau, q\rangle)$ in general, but the differences in relative improvement over the myopic optimization among load patterns is rather small, with steady load having a slight edge over falling and rising load.

Considering that the MLP has the smallest training MSE, it is unexpected to observe that some solutions are worse than the ones that utilize the exponential model. We suspect that the cause for this is attributed to the way in which the training data is generated. More specifically, we intentionally decided to restrict the training data generation to routes in final DYN-$\tilde{t}$ solutions obtained from the ALNS. The reasoning behind that decision is that we want to avoid an overwhelmingly large number of routes that are very bad, to derive finer tuned models for better routes, which usually end up in the solution. This is especially bad for the linear regression and the MLP which are more closely fitted to the training data, whereas the exponential function benefits in this regard from its simplicity and robustness.

Table 7.1: The three discounting approaches, myopic optimization, and sampling applied to ten benchmark instances for each combination of average load and a falling, rising, or steady load as the day progresses.

Load Avg.	Pattern	Approach	$f(\langle R, \tau, q \rangle)$ Median	Change	Trav. time [h] Median	Change	Labor Median	ϕ Median	Change	Runtime [min] Median
10	Falling	Myopic	4057.258	0.00%	67.197	0.00%	136.5	36.125	0.00%	4
		Exponential	3921.867	-2.92%	64.297	-3.14%	424.5	34.920	-3.14%	5
		Linear Regression	**3851.000**	-4.28%	**64.069**	**-4.32%**	**141.0**	35.170	**-4.33%**	8
		MLP	3912.190	**-4.96%**	64.349	-3.92%	147.9	**34.404**	-3.92%	11
		Classical Sampling	3928.320	-4.20%	64.634	-3.80%	201.4	35.157	-3.79%	2950
	Rising	Myopic	3816.300	0.00%	63.413	0.00%	172.5	35.485	0.00%	4
		Exponential	3695.408	-3.49%	61.564	-3.32%	**37.0**	33.920	-3.31%	5
		Linear Regression	**3662.175**	-3.57%	**61.036**	-3.59%	117.5	34.505	-3.58%	8
		MLP	3701.557	**-3.89%**	61.693	**-3.79%**	61.6	**33.976**	**-3.81%**	12
		Classical Sampling	3749.525	-1.14%	62.464	-0.85%	45.5	34.845	-0.85%	3347
	Steady	Myopic	3984.472	0.00%	66.040	0.00%	**0.0**	35.475	0.00%	5
		Exponential	3891.581	-4.31%	64.456	-5.01%	**0.0**	34.000	-5.00%	5
		Linear Regression	3938.683	-4.97%	65.258	-5.63%	135.0	33.795	-5.63%	7
		MLP	3845.711	-5.54%	63.349	-5.48%	**0.0**	33.064	-5.48%	8
		Classical Sampling	**3769.011**	**-8.56%**	**62.771**	**-8.41%**	3.5	**32.618**	**-8.41%**	2543
20	Falling	Myopic	7142.592	0.00%	118.993	0.00%	77.0	32.015	0.00%	24
		Exponential	6802.900	-5.90%	112.105	-5.87%	19.0	**30.025**	-5.87%	32
		Linear Regression	6823.300	-5.40%	112.755	-5.58%	25.0	30.265	-5.57%	49
		MLP	6884.071	-5.10%	112.935	-5.10%	**1.0**	30.360	-5.10%	51
		Classical Sampling	**6693.054**	**-6.32%**	**111.639**	**-6.18%**	4.0	30.127	**-6.18%**	30372
	Rising	Myopic	7027.803	0.00%	116.848	0.00%	380.0	32.365	0.00%	23
		Exponential	6482.008	-6.51%	107.817	**-6.53%**	177.0	**30.965**	-6.53%	32
		Linear Regression	**6419.892**	**-6.62%**	**106.955**	-6.52%	136.0	31.055	**-6.54%**	40
		MLP	6432.858	-6.54%	107.044	-6.22%	124.2	31.065	-6.20%	62
		Classical Sampling	6641.478	-3.79%	110.440	-3.72%	**102.0**	32.138	-3.72%	33728
	Steady	Myopic	7101.992	0.00%	117.963	0.00%	252.5	32.690	0.00%	26
		Exponential	6765.575	**-6.77%**	112.431	**-6.60%**	137.5	31.465	**-6.59%**	28
		Linear Regression	6830.842	-4.02%	113.435	-3.86%	127.5	31.585	-3.85%	37
		MLP	**6721.294**	-5.68%	**111.578**	-5.53%	95.1	31.519	-5.54%	52
		Classical Sampling	6735.598	-5.93%	112.078	-5.60%	**94.0**	**31.060**	-5.60%	25646
30	Falling	Myopic	10432.136	0.00%	172.496	0.00%	**487.5**	31.150	0.00%	77
		Exponential	**9657.147**	**-7.07%**	**159.930**	**-7.01%**	681.0	29.030	**-7.00%**	95
		Linear Regression	9721.894	-5.92%	160.721	-5.55%	713.0	**29.025**	-5.55%	135
		MLP	9689.704	-5.95%	160.690	-5.44%	714.7	29.072	-5.45%	176
	Rising	Myopic	10313.425	0.00%	170.299	0.00%	1308.5	31.055	0.00%	76
		Exponential	9687.325	-6.66%	160.255	**-6.66%**	802.5	**28.690**	**-6.68%**	99
		Linear Regression	9766.358	-6.26%	162.100	-6.14%	**385.5**	29.150	-6.14%	134
		MLP	**9599.087**	**-6.68%**	**159.226**	-6.06%	569.4	29.196	-6.04%	146
	Steady	Myopic	10378.903	0.00%	171.450	0.00%	867.5	31.460	0.00%	82
		Exponential	**9633.436**	**-6.94%**	**159.578**	-6.64%	629.0	**29.225**	**-6.62%**	76
		Linear Regression	9772.233	-5.61%	161.868	-5.32%	**305.5**	29.895	-5.34%	112
		MLP	9802.089	-4.79%	162.408	-4.82%	675.5	29.394	-4.83%	156
40	Falling	Myopic	12632.717	0.00%	209.611	0.00%	508.5	29.530	0.00%	149
		Exponential	**11713.483**	**-7.83%**	**194.497**	**-7.90%**	247.0	**27.420**	**-7.88%**	193
		Linear Regression	11970.428	-6.76%	198.876	-6.56%	**241.5**	27.465	-6.57%	295
		MLP	12031.577	-6.41%	199.944	-6.38%	414.8	27.629	-6.36%	425
	Rising	Myopic	12837.467	0.00%	212.832	0.00%	969.5	30.170	0.00%	195
		Exponential	**12005.597**	**-6.65%**	**199.567**	**-6.58%**	494.5	**27.675**	**-6.57%**	234
		Linear Regression	12238.508	-6.36%	203.612	-6.21%	**408.5**	28.025	-6.20%	311
		MLP	12042.505	-6.57%	199.835	-6.22%	615.2	27.862	-6.21%	332
	Steady	Myopic	12635.717	0.00%	209.439	0.00%	883.0	29.335	0.00%	178
		Exponential	**11715.214**	**-8.04%**	**194.479**	**-8.16%**	540.5	**27.000**	**-8.16%**	170
		Linear Regression	11798.203	-6.49%	196.503	-6.46%	**407.0**	27.560	-6.48%	259
		MLP	11836.576	-7.42%	196.776	-7.45%	535.3	27.341	-7.46%	309

7.6 Conclusions and Future Work

We considered a same-day delivery problem in which dynamically arriving orders have to be delivered within a short time while minimizing travel times, labor costs, and tardiness. We focused on incorporating stochastic knowledge into the objective function of the point-in-time optimization runs, realized by an ALNS, by discounting route durations in dependence on diverse features. The most important features are the number of orders that can be expected up to the latest time the route would need to be started, and the route's mean delivery duration, but several other factors were also considered and partly showed significant benefits.

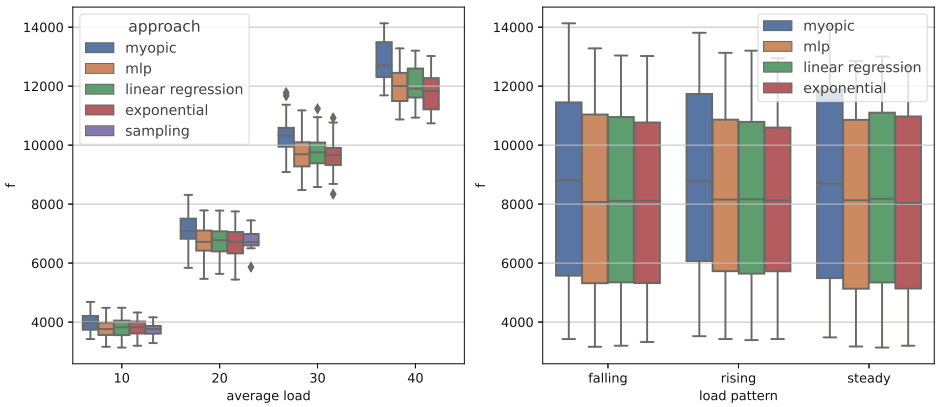


Figure 7.3: Solution quality $f(\langle R, \tau, q \rangle)$ over average loads and different load patterns. The sampling approach is not included in the load pattern graphic due to missing data for average loads greater than 20.

Overall, our experiments clearly indicated that this approach is able to alleviate to a substantial degree the weaknesses of a myopic optimization, in particular in higher load situations. Of the three route duration discounting models, the exponential function performs the best, reducing the travel time as well as the total objective by ≈6.1% on average over all instance classes. The more flexible neural net, in contrast, performed significantly weaker. We conjectured that the reason for this at the first glance surprising observation is the bias we have in the training data. The simpler exponential function seems to be more robust concerning candidate routes with properties that do not appear so frequently in the routes determined by the ALNS when generating training data. Moreover, the independent consideration of the routes is another source of potential errors. In our experiments, the exponential discounting even outperformed the sampling approach regarding solution quality in most cases and cuts down on runtime by several orders of magnitude.

Further work should consider alternative ways of generating training data to possibly reduce the bias. For example, intermediate solutions of the ALNS may occasionally also be used for data generation. Bootstrapping $\hat{\phi}$ from previous non-myopic runs could improve the accuracy of the parameter and lead to further improvement. Moreover, the variability of this mean order duration over the day due to varying load and traffic should be considered. Also, further tests with real-world inspired spatial order distributions (e.g., clustered instances) and load patterns and time-limited optimization could be helpful to evaluate practical aspects of the discounting models.

CHAPTER 8

Towards Learning Value Functions for Same-Day Delivery Problems

In the previous chapter, we have shown how the same-hour delivery short-horizon routing can be substantially improved utilizing surrogate-based optimization for cases when sufficient drivers are available—the *zero-tardiness regime*. In contrast, we focus in this final chapter on the *tardiness regime*, i.e., where tardiness is unavoidable since we have insufficient driver capacities. As stated multiple times, a well-known approach from the literature to increase performance compared to myopic optimization is by sampling and optimizing scenarios in the short horizon and deriving a consensus solution from the resulting plans. Its drawback is the computational effort required, which may not make it suitable for near real-time decision making. To overcome this, we replace this online sampling with an offline training of a short-horizon value function using a neural network, which is then used in the online point-in-time optimization, combining the current reward plus the estimated future value of a solution candidate. This is similar to the surrogate-based approach from before, with the crucial difference that this time we cannot delay the routes' starting times further, without introducing additional tardiness in almost all cases. The ALNS is now primarily guided by tardiness and only as a secondary objective by the travel durations. In a first preliminary computational study on a single-vehicle instance class, we show that the value function approach leads to comparable performance as the sampling approach, while greatly reducing the online decision time.

This work has been presented at the EUROCAST 2022 metaheuristics workshop and accepted for publication in its proceedings:

> Nikolaus Frohner and Günther R Raidl. Learning value functions for same-day delivery problems. In *Computer Aided Systems Theory – EUROCAST 2022*, LNCS. Springer, 2022. accepted

8.1 Introduction

In recent years, same-day delivery problems [162] have gained much attention due to the exploding demand for the fast delivery of goods sparked by the COVID-19 pandemic. They are dynamic vehicle routing problem variants with stochastic customers and have the goal to satisfy customer demand subject to short deadlines efficiently. The orders are mostly unknown in advance and are dealt with upon their availability, hence the route plans change frequently throughout the day, as opposed to static problem variants. Stochastic information is available as spatiotemporal distribution of those orders, often based on prediction models derived from historical data.

In this work, we consider a problem variant where orders arrive in a delivery area throughout the day and are due within one or two hours upon arrival. As opposed to some other variants in the literature [150, 162], every order has to be served and cannot be rejected or delegated to a third party. This is compensated by allowing deadline violations—tardiness—and imposing a penalty upon them. The goal of our problem variant is to minimize a given expected tardiness-dependent penalty function, which is a parameter of the problem. We, therefore, call it the *Same-Day Delivery Problem with Tardiness Penalty* (SDDPTP), an SDDP with soft deadlines as studied by Ulmer [147]. It is a modified variant of [55] as presented in Chapter 5, focusing on the tardiness aspect, which we formulate as route-based Markov decision process in Section 8.2.

For many different dynamic and stochastic vehicle routing problems [126] a decrease in costs/increase in service level is observed when properly including information about future orders over using solely myopic optimization. A general method to deal with those is based on sampling multiple scenarios consisting of real and sampled orders, solving them, and deriving a consensus solution from the different scenario solutions [11, 162]. A drawback of this online method is the high computational demand and resulting runtime at each decision epoch.

For faster decision making, the computational effort is moved to an offline learning phase, where helpful functions are trained to guide the online optimization. In a previous work by Bracher et al. [23] upon which we build, the scenario sampling is only used in an offline phase to train a faster surrogate function to be used in the point-in-time optimization estimating the future mean travel time of routes that can still be delayed. We propose a similar approach in Section 8.4, with the main difference of estimating the tardiness penalty that will accrue in the short horizon when a specific route is started by a driver.

Section 8.5 contains our computational study, where we describe the generation of the SDDPTP instances, how we derive the training and test data for the training and evaluation of our machine learning model, and subsequent performance evaluation on unseen full-day test instances. We restrict ourselves to the single vehicle case with a constant load pattern and compare our value function approach with myopic optimization and state-of-the-art scenario sampling. We observe a new promising routing strategy emerging that prefers shorter routes and early depot returns. It leads to substantially reduced tardiness on both training and unseen test data when compared with myopic

optimization and comes close to the computationally much more expensive scenario sampling approach.

8.2 Problem Formalization

In this section, we describe the SDDPTP in detail and model it as a Markov decision process following the route-based modeling of [147], Ulmer et al. [150] and Voccia et al. [162], where we make suitable adaptions for our problem variant.

We are given vehicles $u \in U$, $|U| = m$ with the respective drivers' shift start times q_u^{start} and respective end times q_u^{end}. Customers are identified as points in a service area $\mathcal{A}$, within which also the depot $\mathcal{D}$ is located. The gross travel time between two orders C_1, C_2 (or one order C_1 and the depot $\mathcal{D}$) is denoted as $d(C_1, C_2)$. The stop times at the customer are denoted as ζ^{c}, the loading time at the depot when starting a route as ζ^{d}, the manipulation time when returning to the depot ζ^{r}. All those times are assumed to be constant and included in the travel time function d.

Orders arrive at times $t^{\text{arr}}(C_i)$ within a planning horizon $[0, H]$ and become immediately available for loading. The deadlines are either one hour or two hours after arrival following a binomial process, given the probability of a one-hour order p_1, and for two-hour orders, $p_2 = 1 - p_1$. No orders are known in advance, all orders have to be delivered by performing multiple unalterable tours, where starting a trip after the assigned driver's shift end time q_u^{end} is not allowed, but finishing is. If after the final return of the last vehicle, there are still orders left, the solution is not feasible. All tours start and end at the depot.

The decision epochs $k \in \{0, \dots, K\}$ are triggered by either the vehicles' returns to the depot at times $t(k)$ or when they are ready again after having decided to wait in the depot. The process consists of pre-decision states S_k, post-decision states S_k^x, and transitions from the latter to the earlier. By definition of our decision epochs, we know that at least one vehicle is in the depot. The set of known orders ready for delivery is denoted by $\mathcal{C}_k$. Each order is included in exactly one driver's feasible tour, which is of the form:

$$\theta_{u,k} = (\mathcal{D}, C_{y_u(1)}, \dots, C_{y_u(l_{u,1})}, \mathcal{D}, C_{y_u(l_{u,1}+1)}, \dots, C_{y_u(l_{u,2})}, \mathcal{D}, \dots, \mathcal{D}), \tag{8.1}$$

with possible multiple returns to the depot. The assignments of the orders to vehicles are denoted by $y_u(o)$ where o is the position in the planned feasible tour and the $l_{u,j}$ is the number of orders of the j-th route, its length. The drivers' return times to the depot are denoted by $\boldsymbol{\rho} = (\rho_u)_{u \in U}$. The pre-decision state is hence a tuple $S_k = (t(k), \boldsymbol{\theta}_k, \boldsymbol{\rho}, \mathcal{C}_k)$, where $\boldsymbol{\theta}_k$ only contains the orders from $k - 1$.

Each pre-decision state contains all relevant information to determine further routing plans and induces a set of possible decisions $x \in \mathcal{X}(S_k)$. A decision deals with planning feasible routes for remaining orders $\mathcal{C}_k$ and deciding, whether to start the first one of the planned routes now or wait for a time Δ in the depot, where the drivers could take a short break. Let $\tau(\boldsymbol{\theta}_k)$ be the *raw* tardiness related to the planned routes, which is used in the tardiness penalty function supplied as a parameter to the problem. For

each order, C_i, $a_i(\boldsymbol{\theta}_k)$ denotes the completion of the delivery (assumed to occur after half the stop time) when the plans $\boldsymbol{\theta}_k$ would be executed as-is starting from the current time $t(k)$ or from the earliest depot return time $\rho_u > t(k)$. This incurs a raw tardiness $\tau_i(\boldsymbol{\theta}_k) = \max(0, a_i(\boldsymbol{\theta}_k) - t_i^{\text{due}})$ per order. We consider the linear tardiness penalty function $\xi^{\text{linear}} = \sum_{i \in \boldsymbol{\theta}_k} \tau_i(\boldsymbol{\theta}_k)$, where with slight abuse of notation $i \in \boldsymbol{\theta}_k$ denote the orders by their indices in the route plan.

We set the (in our case to be minimized) reward $R(S_k, x)$ to the difference in overall tardiness penalty between the plans, i.e., the increase of tardiness in the plan:

$$R(S_k, x) = \sum_{i \in \boldsymbol{\theta}_k^x} \xi_i(\boldsymbol{\theta}_k) - \sum_{i \in \boldsymbol{\theta}_k} \xi_i(\boldsymbol{\theta}_k^x). \tag{8.2}$$

The post-decision state is then $S_k^x = (t(k), \boldsymbol{\theta}_k^x, \boldsymbol{\rho}, \tilde{\mathcal{C}}_k)$, where $\boldsymbol{\rho}$ are the drivers' return times at the depot and $\tilde{\mathcal{C}}_k$ the remaining—planned, but not yet started to be served—customers. Two principal actions per vehicle in the depot are possible: In case we decide to start a driver's first route with customers $\mathcal{C}_{u,k}^1$, they are excluded from $\tilde{\mathcal{C}}_k = \mathcal{C}_k \setminus \mathcal{C}_{u,k}^1$, while ρ_u is set to $t(k) + \bar{d}(\theta_{u,k}^1)$, with $\bar{d}(\theta_{u,k}^1)$ being the duration of the first route in driver u's plan. If we decide to let the vehicle wait in the depot, then ρ_u is set to $t(k) + \Delta$. Note the action space explosion, since we have to select from the set of ordered subsets of all the remaining customers and assign them to the drivers.

From post-decision to pre-decision state a stochastic transition is performed, in terms of realized orders $\mathcal{C}_{\text{r}}(k+1)$ becoming available between $t(k)$ and $t(k+1)$. This updates the set of remaining orders from the previous post-decision state $\mathcal{C}(k+1) = \tilde{\mathcal{C}}(k) \cup \mathcal{C}_{\text{r}}(k+1)$. Furthermore, the return times $\boldsymbol{\rho}$ and the route plans $\boldsymbol{\theta}_k^x$ are carried over excluding the started routes. The initial state S_0 is:

$$(\min_{u \in U} q_u^{\text{start}}, ((\mathcal{D}, \mathcal{D})_1, \ldots, (\mathcal{D}, \mathcal{D})_m), q^{\text{start}}, \mathcal{C}_0), \tag{8.3}$$

where we have an empty route for each vehicle and the orders $\mathcal{C}_0$ that have already arrived before the first vehicle has started its shift. The final state S_K is either at q_u^{end} of the last vehicle or at the last depot return of a vehicle after that, without a route plan and the customers left (ideally none), assuming that there is always one driver shift end after H.

The goal is to find an optimal policy $\pi^* \in \Pi$ so that the expected reward starting from the initial state is minimized:

$$\pi^* = \arg\min_{\pi \in \Pi} \mathbb{E}\left[\sum_{k=0}^{K} R(S_k, X_k^\pi(S_k)) | S_0\right], \tag{8.4}$$

where $X_k^\pi(S_k)$ is the decision rule, selecting a decision when in state S_k according to policy π. The related Bellman equation for the values of states under an optimal policy is given by

$$V(S_k) = \min_{x \in \mathcal{X}(S_k)} \{R(S_k, x) + V(S_k^x)\}. \tag{8.5}$$

Table 8.1: First two epochs with myopic strategy on example instance.

$t(k)$	S_k	S_k^x	$R(S_k, x)$
3600	$((\mathcal{D}, \mathcal{D}), \{1, 2, 3, 4\})$	$((\mathcal{D}, 1, 3, 2, 4, \mathcal{D}), (7607), \{\})$	-6.12
7607	$((\mathcal{D}, \mathcal{D}), \{5, \ldots, 14\})$	$((\mathcal{D}, 6, 13, 5, 11, 8, 10, 7, 14, 12, 9, \mathcal{D}), (15829), \{\})$	-207.77

Table 8.2: First two epochs with sampling strategy on example instance.

$t(k)$	S_k	S_k^x	$R(S_k, x)$
3600	$((\mathcal{D}, \mathcal{D}), \{1, 2, 3, 4\})$	$((\mathcal{D}, 1, 4, 3, \mathcal{D}, 2, \mathcal{D}), (6058), \{\})$	-6.12
6058	$((\mathcal{D}, 2, \mathcal{D}), \{2, 5, \ldots, 12\})$	$((\mathcal{D}, 6, 5, 2, 11, 8, 10, 7, 9, 12, \mathcal{D}), (13205), \{\})$	-47.18

With the state and action space explosion, there is no hope to solve this equation in practice exactly by backward induction known from exact dynamic programming and we have to make use of approximate/heuristic methods.

8.3 Example Instance†

We consider an example instance with $\omega = 4$ hourly orders to arrive following a Poisson distribution for time frame H of eight hours, starting from relative time 0 to $8 \cdot 3600$ seconds, with one-hour order frequency of 0.4, two-hour order frequency of 0.6. The depot is central and the order positions follow a uniform distribution on the unit disc, where drivers move at a constant pace of 15 minutes per unit. One driver is available, with a shift starting at 1h, i.e., $t(k) = 3600$, and ending at 15h, so that there is enough time to deliver the orders.

The objective is to minimize the expected linear tardiness. In Table 8.1, we see the pre-decision states, the post-decision states, and the collected rewards (difference in summed tardiness in minutes from the previous route plan) for the first two epochs using a baseline myopic approach which minimizes the tardiness of the currently available orders as the primary objective and the travel time as secondary. The routes are always started immediately without waiting (except when there are no orders available), which we call *earliest* strategy.

This is compared with a sampling approach in Table 8.2, optimizing scenario plans at each epoch looking two hours ahead and selecting a consensus solution among them that occurs with a highest frequency, after having removed all sampled orders—the consensus algorithm as proposed by Voccia et al. [162]. The planned routes are also started immediately.

We observe that the myopic approach never delays an order (the minimizing of the travel time always seeks to combine orders in one route, if this does not lead to extra tardiness), while the sampling strategy leaves in the first epoch the non-urgent order 2 behind, which benefits the route performance (about three minutes travel time per order less) and return time in a clever way and leads to reduced tardiness in the second epoch since we return earlier to the depot and fewer orders are available by then. To modify the myopic behavior of greedily taking all orders with substantially less online computational effort is our goal.

8.4 Short-Horizon Value Function Approximation

Due to the infamous "curse of dimensionality" [119], the Bellman equation needs to be solved approximately using forward passes and approximating the state's value function, which represents the expected sum of rewards following a given policy:

$$\hat{V}(S_k) = \min_{x \in \mathcal{X}(S_k)} \{R(S_k, x) + \hat{V}(S_k^x)\} \tag{8.6}$$

Ulmer et al. [150] employ an approximate value iteration scheme with state space aggregation, where the states are mapped to a different, coarse-grained state space representation. Approximate values $\hat{V}$ are stored in a lookup table and updated, which is randomly initialized for all the proxy states and then used to solve (8.6) when playing out a randomly sampled episode in a training phase. For each encountered state, the return is collected and then used to gradually update the values in the lookup table. When this is repeated sufficiently often and the state space aggregation captures the real space sufficiently well, the converged $\hat{V}$ should act as an approximation for V. Van Heeswijk et al. [158] propose to learn a linear value function approximation offline to estimate post-decision state values for a delivery dispatching problem combining the approximation of route distances used in an integer linear program to tackle action-space explosion. Joe and Lau [89] employ a reinforcement learning approach, using temporal difference learning with a neural network instead for the value function approximation to train the $\hat{V}$ offline.

We take a surrogate function approach and follow up on previous work from Bracher et al. [23] training a value function using supervised learning. We do not seek to estimate the $\hat{V}$ until the end of the day but instead restrict ourselves to the value accumulated in the near future denoted as $\hat{V}(S_k^x)|_{t(k)+\delta}$, where δ is the duration of the corresponding short horizon. This is similar to Ghiani et al. [67], who select at $t(k)$ a courier dispatching decision from a restricted set by estimating their short-horizon value through an online sampling procedure. As with scenario sampling approaches, the main assumption is that decisions have the most impact on the near feature, which is deemed sufficient to consider for a substantial improvement of routing strategies.

To generate the training data, we run simulations of full-day instances following our Markov decision process. At every epoch, different decisions are sampled from the decision space and evaluated by sampling and solving a set of scenarios within $[t(k), t(k)+\delta]$ and calculating the average tardiness. These are solved as offline problems with perfect knowledge in the short horizon and therefore in general resulting in an underestimation of the costs of any policy. We use them as labels to train a surrogate model $\tilde{V}(S_k^x; \boldsymbol{w}) \approx \hat{V}(S_k^x)|_{t(k)+\delta} + \sum_{i \in \boldsymbol{\theta}_{k+1}} \xi_i(\boldsymbol{\theta}_{k+1})$ with parameters $\boldsymbol{w}$, where the second term is the tardiness of the remaining orders, which are part of the offline problem. Features are derived from the post-decision state and the instance, like the number of expected orders until $t(k)+\delta$ or the tardiness in the remaining orders' route plan. In this offline phase, the actual decision selected is with probability ϵ the best-ranked decision or one at random (ϵ-greedy exploration).

Since shorter routes lead to more available shift time in the short horizon for the offline problem, we expect that their costs are even more underestimated when compared with longer routes. To mitigate this, we multiply the surrogate model with a correction function $\Gamma(S_k^x, \boldsymbol{w}^\Gamma)$. It is not trained by supervised learning but tuned on the full-day training instances in a second phase. As also done by Joe and Lau [89] using simulated annealing, a new online strategy is created by heuristically searching with ALNS at each epoch for the decision $x \in \mathcal{X}(S_k)$ that minimizes $R(S_k, x) + \Gamma(S_k^x; \boldsymbol{w}^\Gamma) \cdot \tilde{V}(S_k^x; \boldsymbol{w}) - \sum_{i \in \boldsymbol{\theta}_{k+1}} \xi_i(\boldsymbol{\theta}_{k+1})$, where we need to subtract once the known tardiness of the remaining orders since they are both considered in R and $\tilde{V}$.

8.5 Computational Study

We implemented the Markov decision process simulator and an adaptive large neighborhood search (ALNS) routing heuristic in Python 3.9. The details of the latter are described in more detail in [55]. For the neural network training, we used Tensorflow Keras 2.8.[1] All training and test runs were performed on a machine with 2×Intel Xeon Gold 6126 with 12 cores each and 2×Nvidia Tesla P100-PCIE-16GB GPUs.

We consider an instance class with $\omega = 4$ hourly orders to arrive following a Poisson distribution for a time frame of eight hours, starting from relative time 0 to $8 \cdot 3600$ seconds, with one-hour order frequency of $p = 0.6$, two-hour order frequency of 0.4. The depot is central and the orders' positions follow a uniform distribution on the unit disc, where drivers move at a constant pace of 15 minutes per unit on Euclidean geometry. Constant times are $\zeta^{\mathrm{d}} = 3\,\mathrm{mins}$, $\zeta^{\mathrm{c}} = 3\,\mathrm{mins}$, and $\zeta^{\mathrm{r}} = 2\,\mathrm{mins}$. We restrict ourselves to one driver with shift starting at relative time 1h, i.e., $t(k) = 3600$, and ending at 15h, so that there is enough time to deliver the orders.

The instances are designed in a way that tardiness is unavoidable in almost all routes, the eponymous tardiness regime. Hence, starting routes early seems more attractive than delaying, since we are most of the time behind schedule. Therefore, we fix the route starting strategy to *earliest*, i.e., if a route is assigned to a driver, it is immediately started, otherwise (e.g., when no orders are available or another driver serves the orders) the driver waits for a defined time of $\Delta = 5\,\mathrm{mins}$.

We estimate the performance of decisions x by the sum of the current reward $R(S_k^x)$ and its value in the short horizon $\hat{V}(S_k^x)|_{t(k)+\rho}$ at $t(k) + \rho$, $\rho = 2\,\mathrm{h}$. Given the recursive nature of the problem, the value is estimated by sampling 30 scenarios until $t(k) + \rho$ and solving the corresponding offline problems, minimizing tardiness as primary and travel time as secondary objective using the ALNS with 100 iterations—see Section 8.4 for more details.

We create training data by sampling, evaluating, and logging decisions for each encountered epoch on 100 full-day training instances and split training and validation data by 70/30. The myopic and scenario sampling decisions are always included in the sampled decisions, while three additional (more likely worse) decisions are created by perturbing

[1] https://www.tensorflow.org

Table 8.3: Performance evaluation neural network on training and validation data with a feature ablation.

features	RMSE-train	RMSE-val	R^2-train	R^2-val
ω', ξ'	88.3	81.6	0.805	0.822
ρ', ω', ξ'	32.7	32.8	0.973	0.971
d', ρ', ω', ξ'	25.4	25.8	0.983	0.982

Table 8.4: Performance evaluation of different strategies on training and test data, $N = 100$ each.

strategy	data	$\overline{\xi^{\text{linear}}}$ [min]	$\sigma_{\xi^{\text{linear}}}$ [′]	$\overline{t^{\text{dec}}}$ [s]	$\overline{K}$	$\overline{\phi}$ [min]	$\bar{l}$	σ_l
myopic	train	18.9	14.0	0.3	7.3	84.7	5.2	1.9
consensus	train	15.3	11.3	37.7	9.6	64.2	4.0	1.3
value-function	train	18.0	11.7	6.0	12.7	44.1	2.7	0.8
value-function-corr	train	15.4	11.0	6.1	10.2	56.1	3.5	1.0
myopic	test	19.4	16.0	0.3	7.9	83.5	5.1	2.1
consensus	test	16.6	13.3	36.4	10.7	65.7	4.0	1.5
value-function	test	19.0	13.4	5.7	13.3	43.7	2.6	0.8
value-function-corr	test	16.6	13.1	5.3	10.6	56.2	3.4	1.1

the myopic solution by randomly moving a (not necessarily contiguous) subsequence of orders to the second route and optimizing both routes with a local search with exchange neighborhood. With 80% probability, the scenario sampling decision is used, with 20% the decision is randomly sampled to diversify the state space traversal. This results in a total of 3351 post-decision state/estimated value pairs for training and validation together. The creation takes a couple of hours using multithreading for scenario sampling to evaluate decisions and dominates the offline phase runtime. The training runtime of the neural network is negligible within a minute.

We consider four different features from the post-decision state, the earliest relative arrival time of the driver ρ', the expected arriving orders within the sampling horizon ω', and the travel time d' and the tardiness ξ' of the known remaining (= not yet started) orders within the route plan. As machine learning model, we use a fully connected feed-forward neural network with two hidden layers with 32 nodes each, trained with Adam optimizer [90] on the mean squared error, 200 epochs at most. To combat overfitting, a weight decay of 0.01 is applied, together with early stopping (patience of 50 epochs with a delta of 10) monitoring the validation loss. We do not observe overfitting comparing the training and validation root mean squared error and the R^2 in Table 8.3 for different feature combinations, and finally select all four features explaining the most variance.

In Table 8.4, we compare the different tardiness penalties and statistics for the decisions and routes of the two baseline strategies myopic and consensus with the value function approach on training and test data, both consisting of 100 full-day simulations. Without a correction function, we observe that the value function selects substantially shorter routes close to three orders per route, resulting in shorter average route durations in minutes $\bar{\phi}$, the expected bias towards shorter routes. For the training and test data, this

leads only to a slightly reduced linear tardiness as compared to the myopic strategy and reduced standard deviation.

As correction function Γ we propose a step function, where a factor scales up the surrogate model $\tilde{V}$ when the duration of the first route is less than a given threshold, otherwise it is kept unchanged. We tune by selecting the best combination of thresholds from $\{1800, 2700, 3600, 4500, 5400, 6300\}$ seconds and factors from $\{1.1, 1.15, 1.2, 1.25, 1.3\}$ regarding the mean performance on the training data, resulting in 2 700 seconds threshold and a factor of 1.3. The results are shown in Table 8.4 as strategy value-function-corr and lead to longer routes as desired with reduced and less varying tardiness close to the consensus method. A Wilcoxon signed rank sum test on ξ^{linear} reveals that for training and test data the corrected value function strategy is now significantly better with a significance level of 1% than the myopic strategy.

Using the neural network, the time per decision $\overline{t^{\text{dec}}}$ is increased to about 5 seconds, whereas the consensus approach takes over half a minute for a decision. Note that this is a single-threaded comparison—both calculations could be sped up, the neural network by batch evaluation of solutions and the scenario sampling by multithreading.

8.6 Conclusions and Future Work

We have formulated a same-day delivery problem with tardiness as a route-based Markov decision process and proposed a supervised learning approach to estimate the value of routing decisions in the short horizon using a neural network. For a first toy instance class with a single vehicle and constant load pattern, we observed a new promising routing strategy emerging preferring shorter routes and early depot returns. It leads to substantially reduced tardiness on both training and unseen test data when compared with myopic optimization and is close to the computationally much more expensive scenario sampling approach. To evaluate its practicability for real-world scenario settings, future research is concerned with the application on a broad set of instance classes with multiple vehicles, varying and larger load patterns, and different delivery area geometries.

CHAPTER 9

Conclusions and Future Work

In the two parts of this thesis, we have first focused on state space search for static combinatorial optimization problems (COPs) using beam search and weighted A* search and second on heuristic solution approaches for dynamic vehicle routing problems with stochastic customers. The corresponding major contributions are a general, parallel beam search framework for combinatorial optimization and a decomposition approach with learnable components for challenging same-hour delivery problem variants, also having shown their effectiveness in practice.

In more detail, we have proposed a novel state-space formulation for the challenging traveling tournament problem (TTP) and compared a randomized beam search approach guided by different heuristics with a reimplemented state-of-the-art simulated annealing approach from the literature. The new state-space formulation in combination with the beam search is effective in finding competitive solutions in reasonable time. Furthermore, we have studied bounded suboptimal weighted A* search on the same state-space, its behavior over different optimality guarantee levels, and how the number of expanded nodes can be reduced with dynamic variable ordering, duplicate state detection, and stronger heuristics.

From this, we generalized to a parallel beam search framework for COPs on many-core CPU systems and clusters. We implemented it in the Julia language and made it open source, together with the implementation of three concrete example problems, the TTP, the permutation flowshop problem (PFSP), and the maximum independent set problem (MISP), and corresponding results of computational experiments. In large beam width runs with multiple workers on a high-performance computing (HPC) cluster, we have found 13 new best feasible solutions on the CIRC, GALAXY, SUPER, and one NFL instance out of 24 tested challenging benchmark instances.

While the weighted A* search showed promising results, solving TTP instances with 12 teams to optimality or with an optimality gap of less than 1% remains an open challenge.

We believe a fruitful continuation could be a batch (weighted) A* search [2], where the evaluation of many open search nodes is performed on a GPU farm. Strengthening of the bound, e.g., combining a computationally efficient variant of the minimum number of trips lower bound and the proposed JCVRPH bound with additional feasibility checks to detect dead-ends as early as possible is likely necessary since the memory demand remains a major issue. This is addressed by iterative deepening A* from Uthus et al. [155] which exchanges the memory demand by accepting many more fast node (re-)expansions and distributing the work on multiple nodes by subtree splitting. Another interesting method could be bidirectional search [141] with which the schedule is constructed simultaneously from the front and the back.

The speedup for the parallel beam search framework is quite large on shared-memory systems in particular with uniform memory access with a dependency on the concrete problem and instance size. A natural further direction would be to compare and combine it with more sophisticated distributed memory approaches, where every worker has its own part of the search tree and communication is performed to select layer-wise the most promising ones and to perform rebalancing of workload if it becomes necessary. For best-first search, parallelization is more challenging since we do not have this layer-wise traversal for which the distribution of work is quite natural. A similar framework for (weighted) A* search would also be an interesting continuation.

To study approaches to challenging dynamic and stochastic optimization problems, we have introduced a same-hour delivery problem originating from an online supermarket in Vienna, which promises to deliver goods within one or two hours. It is a challenging multifaceted decision-making problem, where the core task is to dispatch drivers on planned delivery routes while minimizing and balancing tardiness to keep customer satisfaction high. As a secondary goal, the delivery costs of the company shall be minimized mainly in terms of fuel and labor time. Driver shifts are planned in advance but also dynamically for a given day since shift endings are somewhat flexible and can be extended by overtime.

We decoupled different complex aspects and tackled them on their own, e.g., the route planning, the waiting strategies, and the overtime handling. For the routing, we employ an adaptive large neighborhood search (ALNS) guided by a lexicographic objective function with first customer satisfaction and second delivery costs. A dual-horizon technique is used, where at every decision point a simplified subproblem is solved to estimate desired overtimes for each driver for incorporation into the objective function of the ALNS. Another decoupled decision concerns the route starting times, i.e., whether we should delay a route and let a driver wait further, in the hope that an improvement becomes possible due to newly arriving orders, or start it immediately, avoiding driver idle time. In a simple yet effective heuristic approach, we start inefficient routes later and efficient routes earlier.

The efficiency of a route is represented by the mean order delivery time, also called route or driver performance. What we can expect to achieve depends both on the load (number of available orders) and the current traffic situation—more load, in general,

allows for more efficient routes since the mean distance between orders in the delivery region becomes smaller and obviously less congestion is also beneficial. We train different machine learning models by supervised learning using simulated and real data, achieving an effective predictor for medium to high-load situations, while the low-load situation with only a few drivers remains, as expected, more difficult to predict due to higher variance. This model is used for shift planning and route starting strategies, but itself depends on the prediction of the load and traffic, which we do not consider in our work.

An extension of the model would be to consider the varying geographical distribution of the orders throughout the day, over which we averaged so far, to increase its accuracy. Furthermore, the planning does not consider variance in the traffic and time-dependency of the travel times so far, i.e., at any given point in time they are assumed to be static.

The different strategies and approaches were tested on real-world inspired artificial instances with constant travel times. We showed that the dual-horizon approach and the waiting time strategy together with the ALNS allow balancing performance along the dimensions of tardiness and costs. Still, the routing in the short horizon can be improved, since the ALNS only considers geographical but not temporal aspects and happily mixes urgent with non-urgent orders, which we addressed in the two last chapters of the main body of this thesis.

To anticipate premature depot returns and with it shorter and more flexible routes, the well-known scenario sampling approach with consensus function can be used. Its drawback is the computational demand since we have to sample and solve a number of scenarios for statistical validity, counteracting the required near real-time decisions throughout a delivery day. We propose an offline learning approach, where the scenario sampling is used in simulations to transfer its approximate behavior into a machine learning model based surrogate function. This function is then incorporated into the objective function of the ALNS. It then implicitly respects stochastic information instead of performing myopic optimization by only considering the current reward and costs, while still being sufficiently fast in the evaluation. We have shown that the routing performance can be substantially increased in the zero-tardiness regime on real-world inspired artificial instances. As a basis for further work, we achieved promising results for the tardiness regime, substantially reducing late deliveries in a single vehicle setting.

We believe the presented surrogate function based approaches, related to the cost function approximation method [120, 109, 151], together with recent advances in machine learning methods, can be suitably adapted for other fast-paced online decision making problems with stochastic information and a high degree of dynamism, in particular in the domain of dynamic and stochastic vehicle routing, dynamic scheduling, and shift planning problems. Further research is necessary to identify the concrete properties such kinds of problems have to share for said approaches to be effective. Since we assumed the stochastic information to be perfect in the sense that it correctly captures the probability distributions and also reduced uncertainties of the plan execution, sensitivity analyses, robustness considerations, and model and simulation refinements are deemed important continuations to further improve real-world performance.

APPENDIX A

TTP New Best Solutions

Over the course of our experiments with beam search, we have found several new best feasible solutions. They were found with parallel beam search runs and large beam widths between 20 and 100 million and a final polishing local search. We have reported them to the RobinX repository.[1] Additionally, we present them in Table A.1, representing the schedule as a sequence of games played round by round. To save space, the games are shown in base32 encoding with flat indices for the enumeration of pairs $((t_1, t_2))_{t_1 \neq t_2, t_1=1,\ldots,n, t_2=1\ldots,n}$, where t_2 is the home team.

[1] https://www.sportscheduling.ugent.be/RobinX/travelRepo.php

Table A.1: New best feasible solutions for instances found with parallel beam search over the course of our experiments.

inst	solution	u	l	Δ [%]
CIRC12	C S 2H 1H 3C 44 3F 13 R 2I 2B 42 12 3G Q 2J 2A 43 1 3H 1C 1S 29 31 2 M 1R 1I 3B 32 3 L 1Q 1J 3A 33 2E O 3T 1K 39 2C 23 2F 14 3U 2U 3O A K 2G 26 2T 3V B I 10 2S 1L 1T 34 J 11 17 1M 1U N 35 19 1N 3K 2K 8 3R 36 18 3J 28 7 F 3S 3I 20 2V 9 D 15 21 2D 30 1D 24 P 1V 2M 3P 2P 1E 25 16 2O 3D 5 2Q 1F 1A 40 2N 6 G 2R 1B 38 41 4 H T 37 22 3N 1O E U 27 3M 3E 3Q 1P V 1G 3L 2L	400	388	3.1
CIRC14	1 45 1D 5E 4L 3G 56 2 44 1F 4K 3R 5G 54 43 S 1E 3Q 5F 4M 55 R N 3P 20 2C 2M 38 2S M 17 3C 1V 2D 2O 3 O 3B 21 2B 2N 37 5 3A 4I 1B 53 4A 5J 4 F 4J 52 49 3T 5I 22 4H T 51 3S 5H 4B C 19 1N 2E 2P 35 3U D 1M 1A 2A 2Q 36 3I 1L G 4V 48 2R 34 3J 5A I 10 50 47 4N 3H 4T H V 46 2L 4O 42 6 23 U 1I 33 41 5K 7 P 24 1H 1T 3F 5L 8 Q 11 1G 1U 2K 4R 4G 4U 13 25 1S 32 4F 3M 2T 12 2I 26 4E 4S 18 3N 2U 2J 3E 4D 5M B J 3O 1K 3D 28 57 A K 5C 1J 1Q 29 4P 9 L 16 5D 1R 27 4Q 2F 5B 15 2V 1P 3K 4C 39 2G 14 1C 30 40 59 E 2H 1O 31 3L 3V 58	616	588	4.8
CIRC16	V I 2V 2I 4V 4J 6U 6I 64 J 1D 2U 2J 4U 4I 6V F 10 1H 2H 40 50 5H 6H E 3B 74 1I 26 3V 5U 5I D 73 3C 1J 25 3U 5V 5J 72 S 6L 3D 24 4D 34 60 4N R 16 6M 4C 2Q 35 6B 2C Q 15 4B 6N 36 6C 55 4 2D 14 1S 5D 44 6D 54 5 5M 1V 1Q 5C 3J 46 7B 6 1U 5N 1R 5B 3I 45 7C 5L H 20 30 3H 4H 6F 71 1 1C 1K 2G 4T 41 5K 6G 56 2S 1A 1M 76 4S 3M 6R 2R 57 1B 1L 4R 77 3L 6S 1T K 2T 75 3K 5T 4K 6T 3 M 66 2B 2L 5R 43 4L 2 L 67 2A 2M 5S 42 4M 1E 65 12 2K 31 5E 52 7G A 1F 13 3E 32 5F 63 70 B U 11 3T 3F 4E 6E 62 C T 3R 2F 29 78 4F 5G G 2E 3S 28 33 6Q 53 7D 3A 4O 1G 27 5Q 6P 4G 7F 9 N 19 4Q 5P 6O 6A 3O 8 O 18 5O 5A 69 39 3N 7 P 17 59 68 2N 38 47 3P 49 58 1N 23 2P 37 7E 6J 3Q 4A 1P 22 2O 79 51 48 6K 4P 1O 21 3G 7A 61	898	846	6.1
CIRC18	1K 14 2A 97 40 4I 67 7S 90 13 1L 29 98 42 4K 53 8D 8V 2M N 1M 96 41 4J 55 8U 8E 7 P 70 7I 5F 35 3N 54 6R 9 O 7H 5E 72 36 3L 64 6S 8 7G 1B 71 2J 5G 3M 63 6T 92 3P 1C 21 2H 85 4F 65 8T 3O 12 1D 1V 2I 32 86 52 8R 1 3Q 20 84 33 3E 5K 8S 9E 2 11 6G 7J 34 3D 4H 78 9D 3 10 18 6H 7K 3V 77 9C 6B I 19 4T 2E 6I 7L 76 6A 91 5B 4R 1A 24 2G 2R 44 69 8F 37 5C 4S 23 2F 4E 45 7A 9H 5 38 5D 22 4D 43 8Q 79 9G 4 M 39 4C 8O 89 6N 7Q 9F 6 L 1J 2P 87 8P 6M 7P 68 5S K 1I 2L 3C 51 88 5N 7R 6U 2N 1H 1S 2K 3U 6L 5O 6C 25 6V 2O 1U 6J 4G 7O 5P 9I B 81 8J 1T 2Q 50 6K 7N 7E A J 8K 60 4V 3I 7M 4L 8G F 8I 1E 5V 4U 3T 3J 4M 5R 6D 26 1F 3R 61 3K 4N 9B 5Q G 6E 1G 1P 3S 2U 9A 5L 7C 7F Q 6F 1R 3B 30 99 8A 7D 49 R 16 1Q 31 8N 56 8B 7V E U 4B 28 8M 62 5I 5A 8C D 4A 5U 1O 8L 5H 48 58 6Q C 5T 94 3A 2D 73 47 5J 7T 4Q 93 15 2C 2V 74 46 6O 7U 80 V 27 95 2T 3H 75 59 5M 8H S 82 1N 2S 3G 4P 57 6P H T 17 83 2B 3F 4O 66 7B	1268	1188	6.7
CIRC20	7O 18 5F 9L 8G 94 50 6F 7G BS 1 5E 9K 8F 3B 4V 95 6G AF 8A 2D 5D 9J 8E 3A 3P 96 6H AG 89 E L 25 2V 39 3Q 4L 6R AH B6 F 2E 1B 26 3R 4M 6Q BK 7K B5 G 4Q 1A 27 35 4K BJ 6S 7L B4 4P 7P 19 5G 93 65 9O BL 7M 8R C U 4R 7R 92 5H 9N 66 8T AL B T 7Q 91 2J 9M 3U 5K 8S BR D S 1S 2K 3E 40 4F 6T BQ B8 17 1R 2L 3D 3V 4G 59 7F BP B7 A4 9I 1P 62 6M 3C 7B 4E 5A 9C 9H A5 1O 6L 63 7A 5I 8I 5B 86 8 12 6K 2G 64 8H 5J B3 87 AM A 13 2F 22 3O BH 8J B2 9A 88 9 14 32 24 3N 4H BI B1 9B 9V 6I 31 1N 23 4T BG 4J 5T 7H A0 5V 6J 1M 4S 34 AT 4I 5U 8N AJ 5C 60 1L 1V 2S 4U AU 74 84 AI J R 61 20 2T AS 3T 81 7I 9D H Q 1K 21 2U BF 3S 9Q 83 7J I P 1I 49 BE 3G 9P 5L 70 8O 8U 47 1J BD 2I 3H 51 AE 6V 9T 3J 8V 48 1T 3F 7D AD 6C 6U B9 AN 8C 90 1U 7S 45 7C AC 5M 6E 3 16 8D 4A 7T 44 53 7E 6D A1 8B 15 1F 29 2M 4B 7U 57 72 BM K 1G 28 2O 38 4C 56 73 BN AK 6 AO 1E 2A 2P 36 5N 98 71 BO 7 N 77 AQ 37 43 97 6B 8M A3 5 M AP A8 42 7V 52 8K 99 7N 75 O A6 2B 2N 41 4O 80 5R 8L 46 BB 1C A7 2Q 6N 54 5Q 82 A2 BA 3K 1H 33 2R 4N 55 5S 6A 8Q 4 V 1D 3M 3I 58 B0 69 85 9E 30 10 3L 2C 79 AB AV 68 9S 8P 2 11 78 AR A9 6O 4D 5O 9R 9G 1Q 76 BC 2H AA 6P 67 5P 9U 9F	1724	1600	7.7
SUP12	1O 1E T 26 42 3D 6 3R Q 1R 3B 32 3 O 3U 1T 2N 31 3F G 1F 17 2M 33 23 D 3I 1H 2O 3C 7 J R 1C 38 3L 4 35 3S 1B 1V 29 1 2R 18 27 3V 3O 1D 24 2G 37 2U 3P C 11 1G 2J 3A 3N B E V 1I 21 2L 8 H 10 1A 1S 41 2 1P 2S 2I 2B 44 3Q 2F 36 15 3K 2V 2P 13 3H 1J 39 2D N I 16 1M 2A 3E 9 F P 22 2C 30 2E 3G S 3T 2T 20 34 2Q 1Q 2H 3J 40 5 L 14 1K 2K 43 A K 25 19 1N 1U 12 M U 1L 28 3M	458810	453860	1.1
SUP14	4T 1M 15 2I 29 34 4F 3M 23 T 47 32 3J 5M 5A 44 10 3C 26 36 41 5 F 1E 1U 3F 5J 4R 2 I 1K 1R 2O 55 4P B L 17 1H 51 3R 2L 43 4U U 2V 2C 2R 3U 2S S 46 5E 27 3K 40 39 Q 1N 4J 2A 49 37 C N 1A 1P 4M 5H 3H 4G K 2H 1C 1S 42 57 4 3N 12 1I 2K 35 59 D G 24 3Q 2M 54 4D 7 J 5C 1J 1V 3E 4C 18 M 11 1Q 5F 4O 4E 2F 2T 14 1O 4L 5I 56 A 3A 4I 3P 30 2E 53 1 45 50 3D 28 3S 5L 8 O 2U 1G 20 2B 5G 1L P V 5D 2N 33 4Q E 4V 1D 1T 31 3I 4S 3 4H 3O 2J 52 4A 3L R 2G 25 21 3G 3V 58 22 H 16 1F 2P 3T 5K 6 19 13 4K 2D 38 4B 9 5B 3B 1B 48 2Q 4N	567891	557354	1.9
GAL12	N 2F 16 38 2A 3P 2E O 1C 27 3K 3C C 2G 1B 2T 22 3A 3 D 1L 21 2V 41 7 K 1F 3T 2U 3L 3Q 1E R 2H 2B 3N 2P 13 11 3J 39 29 4 1P 10 26 30 3E 6 M 1Q 2S 1J 3D 8 L 3S 1G 20 2K 1D G 3H 17 2L 43 3F 3R V 19 2I 28 5 3G 2R 15 2D 3B B 2Q Q 37 1U 2C A I S 1R 1N 31 1O E U 1M 40 2M 1 25 18 1H 42 3O 23 35 T 3I 3U 1V 34 24 P 1S 3M 33 12 F 36 1T 2O 32 9 J 14 1I 3V 2N 2 H 1A 1K 2J 44	7135	7034	1.4
GAL14	E 10 1B 3R 38 55 4D 2S 23 U 1F 53 40 5K 22 2T 12 1D 51 3V 5L 1 1N 4J 5F 2L 3H 4E 1L G V 2R 4N 3K 4C 43 K 3B 2I 3Q 52 4S 6 Q 1A 30 2B 3U 58 4 19 5C 2C 3S 33 57 8 4H 16 1O 48 32 5J 4G 44 13 3C 1R 2K 5M 3 I 45 21 2P 3G 41 7 O 4V 1G 26 2O 3L D N 15 1J 1Q 29 34 18 5B 14 1S 27 37 4P 5A 3N 2U 46 1P 2M 4R 3M 3A 4I 50 2J 2E 4A B J 3O 25 3D 54 4F A S 1I 1T 28 3F 59 5 M 2H 1H 20 36 5I R P 1C 5E 2N 35 4O 4T F 3P 4K 31 5G 4B 39 4U 24 1E 47 4M 42 2 1M 1K 2D 49 3T 3J C 2G 17 2V 1V 2A 3I 9 L 11 5D 1U 4L 2Q 2F H T 3E 5H 56 4Q	10840	10255	5.7
GAL16	3 J 15 2H 5T 4K 5G 71 7 N 1B 1N 2G 5R 4M 6U D L 1D 1O 5P 2J 4F 6T 3A 3Q 66 1I 25 38 51 7E 3P 65 3C 4Q 2A 5B 36 7B 64 3B 3R 59 28 4S 33 7G 8 S 12 5O 2Q 6P 5D 4H 4 U 14 1Q 2L 5S 6R 4I 6 49 16 1S 6N 2M 3J 61 1T 73 1G 30 3K 41 52 6H 72 I 1V 3F 5C 44 6D 54 5 M 74 2U 5A 46 50 6G 1E 2D 18 4C 31 5E 63 70 2C 1F 19 76 37 6Q 4G 5V 1 2E 6M 24 78 3I 6E 60 2 R 4B 4R 77 3V 3N 6F 48 T 58 20 3U 4T 3L 6I 56 O 13 1K 29 39 4V 6V F 57 11 23 2P 34 6C 5U A 10 3D 68 2K 35 7A 62 B 1U 6L 67 2N 4E 40 7C 6J H 1P 2V 4D 4U 45 7D G 4A 1R 21 3G 47 53 5H V 5M 1M 3E 2O 32 55 5J E P 5N 2F 2B 6A 3H 5F C Q 2T 3S 26 69 79 6S 2R K 1C 75 27 6B 42 4J 4N 2S 1A 1H 2I 43 4L 7F 5L 4O 17 1L 22 6O 3M 5K 9 6K 4P 1J 3T 5Q 3O 5I	14583	13702	6.4
GAL18	I 94 1Q 2A 30 52 5P 6S 7S 92 J 20 84 2V 50 7M 6L 6B 2 93 7I 4U 35 86 75 64 5M 37 14 71 3S 85 4K 8Q 5O 6C 13 38 1U 29 48 4L 5K 6R 8V 2M U 39 1T 7J 40 4N 6O 91 4 O 2O 22 7L 9A 54 6N 7D 6 K 1I 96 7K 87 76 53 67 5 6V 1A 83 2G 98 5I 59 69 8H 4A 1J 5E 2I 3T 3I 65 8D 49 1L 8J 2D 34 3K 42 5N 6T 1K L 4B 32 3F 41 6A 8T 7V F N 1M 72 2S 7N 55 5L 9I A M 16 3R 74 4H 9D 8C 7U B 26 18 6G 73 8O 4M 89 9C 25 4R 17 3A 6K 4P 66 8U 8E 4Q S 27 21 2T 3D 5R 6P 90 5S 10 4S 24 2Q 3J 47 5J 7R 1 5U 1V 2J 2R 3V 9B 5Q 7B 80 5T 15 2E 6I 3U 4O 77 9G 3 81 19 2F 8M 6J 4I 57 7E 7 Q 1F 8K 6H 3C 4J 7P 8G D 8I 5D 1R 3B 2U 46 7Q 9E 5B 6E 1B 23 60 36 3E 44 7T 6D 5C 1G 1O 2L 62 51 8P 7C 3O 12 6F 4T 8L 33 4F 8A 79 9 V 3Q 95 4D 31 8N 78 8B 8 T 1C 2P 2K 3L 99 7O 68 G 7G 1H 1S 2C 5G 3N 63 7A 6U 3P 1D 1N 4E 5H 5A 6Q 8F H 2N 70 1P 5F 45 88 6M 8S E P 82 28 97 3H 43 56 8R C 11 7H 4C 2B 61 3G 58 9F 7F R 1E 5V 2H 4V 3M 4G 9H	20205	19051	6.1
GAL20	2 3K 20 2R 8G 55 AD 69 7M A3 H M 1M 2M 38 44 50 99 84 8T 9 Q 1P 4S 2N 3G 3U 98 89 8Q 1Q 10 1E 34 4C 5S BK 6T B5 A0 5V 76 1S 2I A9 4D 8K 73 9U BP 75 60 9J 26 4A 5H 4V 6V B6 AM 1 77 62 3N 93 7V 8J 5U B2 AK 17 V 7R 8F 35 6P 59 5R B1 9G 46 15 61 25 2H 8H 51 6U 9D BR C 11 48 2B 4T 3C 9N BJ 6F 70 I BB 1L 27 9L 36 3T 7D 68 AH E AO 1H 33 5G 3S AB 9P 7N 86 AN 8C 1G A7 2V 3O 95 5K 83 7K 8B 16 AP 91 2S 6N 5I 4M 54 82 BA 13 8D 2A 6M 3B 65 4E 5A 9B D 47 BC 22 63 3H 6O 56 9T AJ 7 1R 1J BE 3D 3R 67 AF 7J A2 3 9I 1D 2K AS 40 5Q AE 7G 8A 8U N 19 4U 43 8I 5M 7F 87 B9 30 P 90 8E 7S 4H 5B 5L 72 B7 K 32 1V A8 4N 58 6R 6D BM 8O F 31 6K 21 2P AA 4J 5N 9R BS A 18 1U 2J 41 AC 5T 9S 7H BO B 7P 3L 2G 9M 7C 96 5P 6H B8 7O 14 4R 78 92 3I 3P B0 6E 8M 4P S 7Q 49 2T 3F 94 8L 74 B3 A4 4Q 1F 23 2U 7A BG 4I 71 8P 4 A5 1K 28 BF 7B 9O 6Q 5O 6G 5 12 A6 9K 79 3Q 5J 6C BL B4 9H U 2F 5F 37 45 AU 6A 85 9E 2D T 1B AQ 42 4O 53 81 9A 8R 6I 8V 1C 1T 4B 52 7E 88 8S BQ 8 6J 1A 3M 39 4K 9Q 7I BN AL 3J 2E 5E 6L 3E BH AV 6B AG 9V 5C L 2C AR 3A 3V 66 57 8N A1 J 5D 1N 29 2O 64 AT 4G 80 9C G O 1O 24 2Q 7U 4F BI 97 7L 6 R 1I BD 2L 7T 4L 6S AI 9F	25401	23738	7.0
NFL22	2L 40 BT 2F 7D DC 5L BF CQ 8B 9R L B7 7A 6M 34 DA 47 C2 5N 89 CU E U D7 2O 3R A2 7F 5K 72 92 D4 K 16 1O AK 62 3F 4R 74 8T E6 AE AH 10 1R 29 43 3S 57 66 9J E7 AD CG 21 1L 4O 82 46 67 9K E9 CB BP 59 6K 7V 28 2P 45 7K 9L D0 BQ DN 5 DR 2N CJ DB 84 5O 6S 7M 93 BK 2 V 5C 2R 3Q 8O E3 7J AA B2 DO 6 T 1D AL 3K 51 6E E4 C7 94 AF 1B P 2I 3J 4A 54 AP 68 C8 9Q AG 7T 5V 1Q B9 9B CL 4B 4U 5T 76 CA D 79 3C 80 36 4D CN 9F DE 77 BM F D6 CI 2D 2Q DV 8P 6R 7H 8C BN C 17 8K 2G 37 4P E0 A4 AQ DF 9I 5U R 1M 2A A0 8N E1 5S AR 9N CD BR 18 1N 61 6N 3H 4C AO BH A8 DP 1 1P BU 3E 4J 50 5I 6D 7L 8D D3 9 5A AJ 2C 39 6O 4I 52 88 9O D1 I 8J 6L 2E 38 AM 9D 4V 5H CR EB 9S Q 1E 2B C1 6Q BE 8R E5 9P DL 78 S 1I D8 BA 3L CM 6F 8S E8 AB D5 N 9V 81 3M 4G 58 5M 6G BG 7N A BS 1V 7B 31 3O 64 9E 5P 8G B3 H CH 1U 26 33 9C AN 5J C4 6V 8H B 3B 1H 27 30 4F CP 71 95 9M EC B6 15 9U 23 3I 49 55 8Q 6U DJ ED 4K O DS 8M A1 48 DD 6B 7P AT BO M 22 32 5E BC A3 6I 87 7Q AU CE 3 7U 1F 35 7E 5G 6H 9H CT B0 CF G 98 5B 8L 2V 3G 6P 86 CV CC EE 6J 9T 4M 2H D9 44 E2 9G 7O CS 91 8I 13 1S 25 DU 3T 53 7G A5 75 BI 20 19 1G 7C 3U 65 5R A6 8E BJ AV 7 1A 60 42 2S 3N C5 AS 8F DI BL 8 1C 9A CK 63 83 4T C6 96 DK B1 4 4L 1J DT C0 4E CO 73 A7 DH 8V 97 2M 1K 24 4H 4S 5Q 7R 8U AC EA DQ 11 99 2J 2T 3P 5F BD 7I A9 D2 J 14 B8 3D 2U 4Q C3 6A 7S 8A B4 3V 12 1T 4N BV BB 85 6C 70 DG B5 3A AI 41 2K 5D 56 69 6T 90 C9 DM	400636	378813	5.8

APPENDIX B

Same-Hour Delivery

B.1 Real-World Inspired Artificial Instances Generation

Artificial test instances have been created as we are not allowed to publish data from the real application. These instances are designed to reflect the most important relationships found in the application. We seek to model the delivery area of a single depot and simulate the varying demand occurring over a day by artificially created instances. Such an instance is primarily characterized by its number of orders n and its number of vehicles m. As a simplified area, we take the unit disk centered in the Euclidean plane where the depot is located at $(0, 0)$. Orders may occur within 16 opening hours from 9am to 1am represented by $H_{\mathrm{O}} = [9, 25]$ and either have a promised maximum delivery duration of one hour or two hours. Our planning horizon, therefore, extends to $H = [9, 27]$.

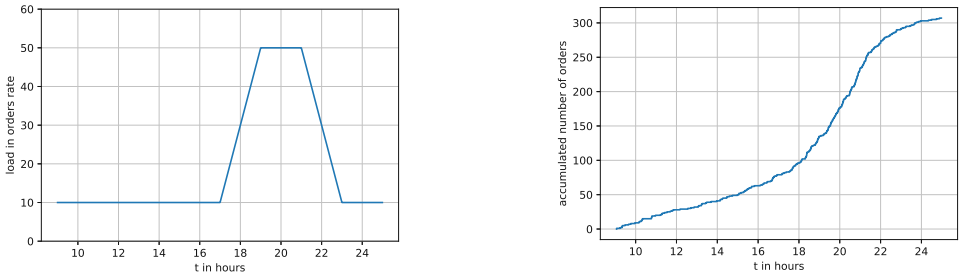


Figure B.1: Left: Load function ω_{BD} with base load of ten orders arriving per hour and trapezoidal peak ($t_{\mathrm{A}} = 17,\ t_{\mathrm{B}} = 19,\ t_{\mathrm{C}} = 21,\ t_{\mathrm{D}} = 23$) with additional 40 orders per hour. Right: Realization of a corresponding inhomogeneous Poisson process displaying the accumulated number of orders over time.

Euclidean distances between orders and the depot are converted to travel times via the constant pace of 20 minutes per unit distance. Furthermore, stop times of three minutes to/from an order, a general manipulation of two minutes when going to the warehouse, and a loading time of 20 seconds when leaving the warehouse are added to the travel times. The picking time $t_v^{\mathrm{rel}} - t_v^{\mathrm{avl}}$ for each order $v \in V$ is assumed to be three minutes.

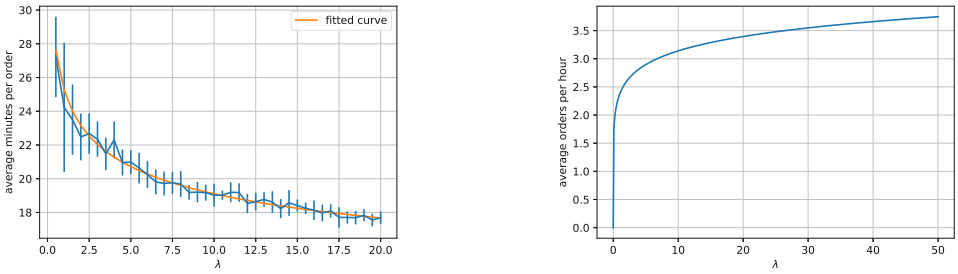


Figure B.2: Left: Mean average minutes per order with standard deviations over ten instances for each load value $\lambda \in \{0.5, 1.0, \ldots, 20.0\}$ with a fitted curve following an inverse proportional power law. Right: Average orders per hour $1/\overline{t_O}$ derived from the fitted curve.

We assume the orders to occur spatially uniformly at random on the disk following an *arriving load* distribution $\omega\colon H_O \to \mathbb{R}_0^+$ where one-hour orders are assumed to be slightly more likely than two-hour orders with a probability of 60%. The load function is composed of a base load L_B which is constant throughout the day and a peak load L_P to which we ascend and from which we descend linearly using a trapezoid with parameters t_A, t_B, t_C, t_D. See Fig. B.1 for an example together with a realization of a corresponding inhomogeneous Poisson process. We define two different reasonable load patterns, one for business days P_{BD}, where the peak starts at 5 pm and one for weekends/holidays P_{WE}, where the peak is more smeared over the day since the load begins to increase already earlier the day:

$$P_{BD} = (L_B = 10, L_P = 40, 17, 19, 21, 23) \tag{B.1}$$
$$P_{WE} = (L_B = 10, L_P = 50, 11, 17, 20, 22) \tag{B.2}$$

For P_{BD} we are in the order of 300 orders per day and for P_{WE} around 500 orders.

The second part of the input besides order-related information concerns the drivers' shifts. How many drivers we need in a given hour depends on the expected driver performance: The number of delivered orders per hour or stated differently the expected labor time needed by a driver to deliver a single order, as simplification ignoring waiting time at the depot. To estimate this, we employ a bootstrapping mechanism since this performance depends on our route construction and optimization and departure time strategy itself. For different but constant loads we solve the dynamic problem for an "infinite" time horizon (to get rid of burn-in effects) given an "infinite" number of vehicles (to not run into late deliveries due to capacity limitations). We conjecture that the probability density function of the number of delivered orders per hour converges to a steady state distribution, from which we can take the mean. For low loads, we expect the performance of the drivers to be worse, since the average distance between orders is larger. If more orders are available, chances seem higher to create better routes with less time needed per order.

We verified this empirically by creating ten instances over an order time horizon of 20 hours for different constant loads to be solved as a DYN problem using ALNS with 60 seconds for route optimization at each arriving order. We let the load vary from 0.5 to

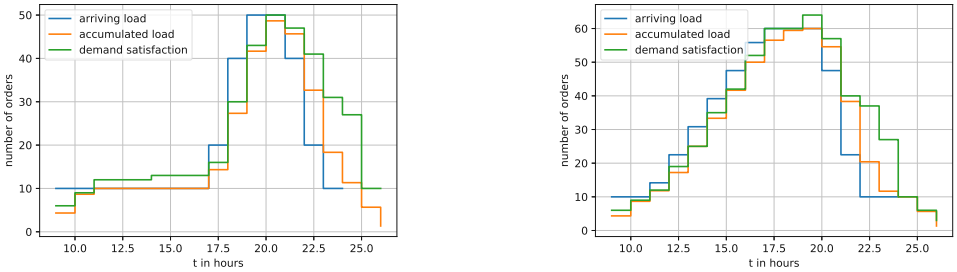


Figure B.3: Discretized business day (left)/weekend (right) load curve (blue) and corresponding lagged accumulated load (orange) giving rise to one random greedy shift planning realization (over-)satisfying expected load (green).

20.0 orders per hour in steps of 0.5. For each dynamic problem instance, this yields many routes and with each a certain duration and a number of orders served. This allows us to calculate the average time per order $\overline{t_O}$ for each of the ten instances over all different loads. As expected, lower loads yield worse performance than higher loads. Given this measurement, we make a parameterized ansatz for the relation between the load and the average minutes per order:

$$\overline{t_O}(\lambda; A_1, \alpha + 1) = A_1 \lambda^{-\alpha_1} + 6\,\text{min} \tag{B.3}$$

The six minutes correspond to the stop time per order which does not vanish even if the travel time goes to zero. Fitting this model to our measurements yields parameters $A_1 = 19.3$ and $\alpha_1 = 0.17$. Figure B.2 visualizes the measured time per order with its standard deviation and the fitted curve on the left side. An alternative and sometimes more convenient representation of this performance is given by the average orders per hour $1/\overline{t_O}$, which we plotted for a larger range of $\lambda \in [0, 50]$ in Fig. B.2 on the right.

Finally, we need to create shifts that allow the expected demand to be satisfied in time. For this, we discretize the load curve into slots of hours and take for each the hourly mean, corresponding to the expected number of orders arriving in this hour $\hat{\omega}(t)$. Due to the different order types (60% with one hour maximum delivery duration vs. 40% with two hours), the arriving load lags and distributes over the current and the subsequent hours, yielding an *accumulated load* $\Lambda(t)$ for each hour. We assume an even split of the load for the one-hour orders over two hours and for the two-hour orders over three hours. For example, when the hour t has an expected load of 30 arriving orders, 13 orders will be assigned to the hours t and $t + 1$ ($60\%/2 + 40\%/3$), and four orders to hour $t + 2$ ($40\%/3$). At the beginning of each hour, the necessary number of shifts is added staggered over the hour to satisfy the accumulated load together with the already existing shifts in this hour. To estimate the performance of the drivers, the average number of orders a driver can serve in this hour depending on the accumulated load is taken into account. Optionally, we close the oldest shifts that are at least four hours long, when we over-fulfill the demand. When adding a driver, the corresponding shift duration is sampled randomly from $\{4, \ldots, 7\}$ hours, capping at the end of the planning horizon. This yields a set of drivers U with shift start and end times, where the earliest shift end time q_u^0 is set to half an hour before q_u^{end}, $u \in U$, to allow for shift ending flexibility in the

dynamic problem. To simulate driver shortage, we optionally drop one shift at random. Figure B.3 shows the load satisfaction on one such randomized greedy shift planning for the business day and weekend load patterns, respectively.

To summarize, to come up with meaningful artificial instances, orders are created uniformly randomly on a Euclidean disk, modeling the delivery area in a simplified way. Orders occur over the planning horizon following either a business day off-peak vs. peak load pattern or a more smeared weekend load pattern to be sampled from an inhomogeneous Poisson process. We measured the dependence of the average orders per hour a driver could perform depending on the accumulated load in that hour and derived an inverse power law function by curve fitting. This enables us to perform a greedy shift planning where we check and if necessary open shifts staggered at every hour to satisfy the accumulated load derived from the expected load patterns. This is done for each hour of the extended planning horizon H, where the durations of newly opened shifts are sampled randomly from $\{4, \ldots, 7\}$ hours. Optionally, we close shifts early when there are more than enough drivers and as a second, cascaded option, drop a random shift in a final perturbation step.

B.2 Additional Results

In Figure B.4, the detailed boxplots for the different solving strategies over the three objective dimensions and six instances classes are shown.

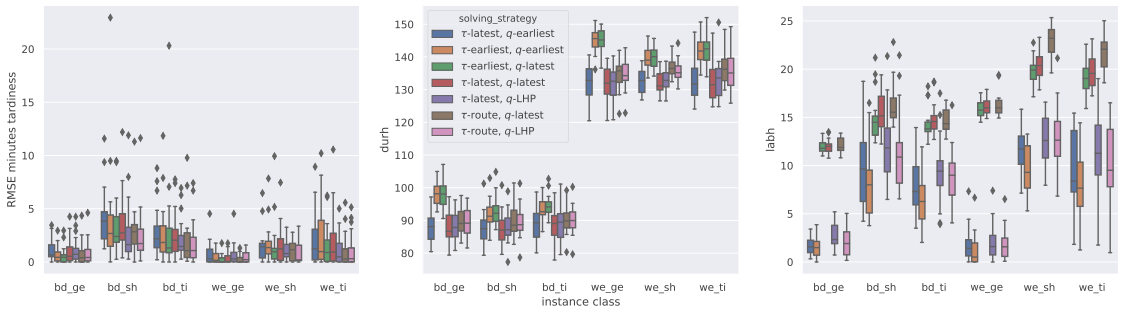


Figure B.4: Comparison of the root mean square error of the tardiness in minutes, the travel time duration, and the excess labor time of different solution strategies (without offline solution) on six different instances classes with 20 instances each. We observe that the more sophisticated strategies based on LHP and the route performance decreases the tardiness at the cost of carefully introducing additional travel time (regarding which τ-latest is best) and labor time (where q-earliest is best).

List of Algorithms

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Forest Agostinelli, Stephen McAleer, Alexander Shmakov, and Pierre Baldi. Solving the rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8):356–363, 2019.

[3] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS '67 (Spring): Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, pages 483–485. Association for Computing Machinery, 1967.

[4] Aris Anagnostopoulos, Laurent Michel, Pascal Van Hentenryck, and Yannis Vergados. A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, 9(2):177–193, 2006.

[5] Diogo V Andrade, Mauricio GC Resende, and Renato F Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18(4):525–547, 2012.

[6] David L Applegate, Robert E Bixby, Vašek Chvátal, and William J Cook. *The traveling salesman problem*. Princeton University Press, 2011.

[7] Nabila Azi, Michel Gendreau, and Jean Yves Potvin. A dynamic vehicle routing problem with multiple delivery routes. *Annals of Operations Research*, 199(1): 103–112, 2012.

[8] Nabila Azi, Michel Gendreau, and Jean Yves Potvin. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers and Operations Research*, 41(1):167–173, 2014.

[9] Jillian Beardwood, J. H. Halton, and J. M. Hammersley. The shortest path through many points. *Mathematical Proceedings of the Cambridge Philosophical Society*, 55 (4):299–327, 1959.

[10] Russell Bent and Pascal Van Hentenryck. Waiting and relocation strategies in online stochastic vehicle routing. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, page 1816–1821. Morgan Kaufmann Publishers Inc., 2007.

[11] Russell W Bent and Pascal Van Hentenryck. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, 52(6): 977–987, 2004.

[12] David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John N Hooker. Variable ordering for the application of BDDs to the maximum independent set problem. In *International Conference on Integration of AI and OR Techniques in Constraint Programming*, volume 7298 of *LNCS*, pages 34–49. Springer, 2012.

[13] David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John N Hooker. Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing*, 26(2):253–268, 2013.

[14] David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John N Hooker. *Decision Diagrams for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, 2016.

[15] David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John N Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1): 47–66, 2016.

[16] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[17] Roberto Bisiani. Beam search. *Encyclopedia of Artficial Intelligence*, pages 1467–1468, 1992.

[18] Christian Blum and Günther R Raidl. *Hybrid Metaheuristics: Powerful Tools for Optimization*. Springer, 2016.

[19] Christian Blum, Maria J Blesa, and Manuel Lopez-Ibanez. Beam search for the longest common subsequence problem. *Computers & Operations Research*, 36(12): 3178–3186, 2009.

[20] Flavia Bonomo, Andrés Cardemil, Guillermo Durán, Javier Marenco, and Daniela Sabán. An application of the traveling tournament problem: The argentine volleyball league. *Interfaces*, 42(3):245–259, 2012.

[21] Nils Boysen, Stefan Fedtke, and Stefan Schwerdfeger. Last-mile delivery concepts: a survey from an operational research perspective. *OR Spectrum*, 43(1):1–58, 2021.

[22] Adrian Bracher. Learning surrogate functions for the short-horizon planning in same-day delivery problems. Bachelor's thesis, TU Wien, Institute of Logic and Computation, 2022.

[23] Adrian Bracher, Nikolaus Frohner, and Günther R Raidl. Learning surrogate functions for the short-horizon planning in same-day delivery problems. In *17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR'21)*, volume 12735 of *LNCS*, pages 283–298. Springer, 2021.

[24] Jürgen Branke, Martin Middendorf, Guntram Noeth, and Maged Dessouky. Waiting strategies for dynamic vehicle routing. *Transportation science*, 39(3):298–312, 2005.

[25] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. hwloc: A generic framework for managing hardware affinities in hpc applications. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pages 180–186. IEEE, 2010.

[26] Aydin Buluç and Kamesh Madduri. Parallel breadth-first search on distributed memory systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2011.

[27] Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. NuMVC: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research*, 46:687–716, 2013.

[28] Diego Cattaruzza, Nabil Absi, and Dominique Feillet. Vehicle routing problems with multiple trips. *Annals of Operations Research*, 271:127–159, 2018.

[29] Tristan Cazenave. Monte carlo beam search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):68–72, 2012.

[30] Jinho Choo, Yeong-Dae Kwon, Jihoon Kim, Jeongwoo Jae, André Hottung, Kevin Tierney, and Youngjune Gwon. Simulation-guided beam search for neural combinatorial optimization. *arXiv preprint arXiv:2207.06190*, 2022.

[31] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[32] Joseph C Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[33] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[34] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.

[35] Carlos F Daganzo. The distance traveled to visit N points with a maximum of C stops per vehicle: An analytic model and an application. *Transportation Science*, 18(4):331–350, 1984.

[36] Dominique De Werra. Scheduling in sports. *Studies on graphs and discrete programming*, 11:381–395, 1981.

[37] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505–536, 1985.

[38] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255, 2009.

[39] Luca Di Gaspero and Andrea Schaerf. A composite-neighborhood tabu search approach to the traveling tournament problem. *Journal of Heuristics*, 13(2):189–207, 2007.

[40] John P Dickerson, Ariel D Procaccia, and Tuomas Sandholm. Dynamic matching via weighted myopia with application to kidney exchange. In *Twenty-sixth AAAI conference on artificial intelligence*, 2012.

[41] Marko Djukanović, Günther R Raidl, and Christian Blum. A beam search for the longest common subsequence problem guided by a novel approximate expected length calculation. In *International Conference on Machine Learning, Optimization, and Data Science*, volume 11943 of *LNCS*, pages 154–167. Springer, 2019.

[42] Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer, 2012.

[43] Cees Duin and Stefan Voß. The pilot method: A strategy for heuristic repetition with application to the steiner problem in graphs. *Networks: An International Journal*, 34(3):181–191, 1999.

[44] Guillermo Durán. Sports scheduling and other topics in sports analytics: A survey with special reference to latin america. *Top*, 29(1):125–155, 2021.

[45] Guillermo Durán, Santiago Durán, Javier Marenco, Federico Mascialino, and Pablo A Rey. Scheduling argentina's professional basketball leagues: A variation on the travelling tournament problem. *European Journal of Operational Research*, 275(3):1126–1138, 2019.

[46] Kelly Easton, George Nemhauser, and Michael Trick. The traveling tournament problem description and benchmarks. In *International Conference on Principles and Practice of Constraint Programming*, volume 2239 of *LNCS*, pages 580–584. Springer, 2001.

[47] Kelly Easton, George Nemhauser, and Michael Trick. Solving the travelling tournament problem: A combined integer programming and constraint programming approach. In *International Conference on the Practice and Theory of Automated Timetabling*, volume 2740 of *LNCS*, pages 100–109. Springer, 2002.

[48] Rüdiger Ebendt and Rolf Drechsler. Weighted A* search–unifying view and application. *Artificial Intelligence*, 173(14):1310–1342, 2009.

[49] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5(1):17–60, 1960.

[50] Ariel Felner, Richard E Korf, and Sarit Hanan. Additive pattern database heuristics. *Journal of Artificial Intelligence Research*, 22:279–318, 2004.

[51] Miguel Andres Figliozzi. Planning approximations to the average length of vehicle routing problems with varying customer demands and routing constraints. *Transportation Research Record*, 2089(1):1–8, 2008.

[52] Miguel Andres Figliozzi. The impacts of congestion on commercial vehicle tour characteristics and costs. *Transportation Research Part E: Logistics and Transportation Review*, 46(4):496–506, 2010.

[53] Nikolaus Frohner and Günther R Raidl. Towards improving merging heuristics for binary decision diagrams. In *Proceedings of LION 13 – 13th International Conference on Learning and Intelligent Optimization*, volume 11968 of *LNCS*, pages 30–45. Springer, 2019.

[54] Nikolaus Frohner and Günther R Raidl. Merging quality estimation for binary decision diagrams with binary classifiers. In *Machine Learning, Optimization, and Data Science – 5th International Conference, LOD 2019*, volume 11943 of *LNCS*, pages 445–457. Springer, 2019.

[55] Nikolaus Frohner and Günther R Raidl. A double-horizon approach to a purely dynamic and stochastic vehicle routing problem with delivery deadlines and shift flexibility. In *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling - PATAT 2021: Volume I*, 2020.

[56] Nikolaus Frohner and Günther R Raidl. Learning value functions for same-day delivery problems. In *Computer Aided Systems Theory – EUROCAST 2022*, LNCS. Springer, 2022. accepted.

[57] Nikolaus Frohner, Bernhard Neumann, and Günther R Raidl. A beam search approach to the traveling tournament problem. In *Evolutionary Computation in Combinatorial Optimization – 20th European Conference, EvoCOP 2020*, volume 12102 of *LNCS*, pages 67–82. Springer, 2020.

[58] Nikolaus Frohner, Stephan Teuschl, and Günther R Raidl. Casual employee scheduling with constraint programming and metaheuristics. In *Computer Aided Systems Theory – EUROCAST 2019*, volume 12013 of *LNCS*, pages 279–287. Springer, 2020.

[59] Nikolaus Frohner, Matthias Horn, and Günther R Raidl. Route duration prediction in a stochastic and dynamic vehicle routing problem with short delivery deadlines. In *Proceedings of the 2nd International Conference on Industry 4.0 and Smart Manufacturing (ISM 2020)*, volume 180 of *Procedia Computer Science*, pages 366–370. Elsevier, 2021.

[60] Nikolaus Frohner, Jan Gmys, Nouredine Melab, Günther R Raidl, and El-Ghazali Talbi. Parallel beam search for combinatorial optimization (extended abstract). *International Symposium on Combinatorial Search*, 15(1):273–275, 2022.

[61] Nikolaus Frohner, Jan Gmys, Nouredine Melab, Günther R Raidl, and El-Ghazali Talbi. Parallel beam search for combinatorial optimization. In *51th International Conference on Parallel Processing Workshop*, ICPP Workshops '22. Association for Computing Machinery, 2022.

[62] Nikolaus Frohner, Bernhard Neumann, Giulio Pace, and Günther R Raidl. Approaching the traveling tournament problem with randomized beam search. *Evolutionary Computation Journal*, 2022. in press.

[63] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[64] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 1995.

[65] Michel Gendreau, Francois Guertin, Jean-Yves Potvin, and Éric Taillard. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation science*, 33(4):381–390, 1999.

[66] Michel Gendreau, Jean-Yves Potvin, et al. *Handbook of metaheuristics*, volume 2. Springer, 2010.

[67] Gianpaolo Ghiani, Emanuele Manni, Antonella Quaranta, and Chefi Triki. Anticipatory algorithms for same-day courier dispatching. *Transportation Research Part E: Logistics and Transportation Review*, 45(1):96–106, 2009.

[68] Matthew L Ginsberg and William D Harvey. Iterative broadening. *Artificial Intelligence*, 55(2-3):367–383, 1992.

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

[70] Fred Glover and Manuel Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.

[71] Jan Gmys. Exactly solving hard permutation flowshop scheduling problems on peta-scale GPU-accelerated supercomputers. *INFORMS Journal on Computing*, 34(5):2502–2522, 2022.

[72] Jan Gmys, Mohand Mezmaz, Nouredine Melab, and Daniel Tuyttens. A computationally efficient branch-and-bound algorithm for the permutation flow-shop scheduling problem. *European Journal of Operational Research*, 284(3):814–833, 2020.

[73] Marc Goerigk and Stephan Westphal. A combined local search and integer programming approach to the traveling tournament problem. *Annals of Operations Research*, 239(1):343–354, 2016.

[74] Marc Goerigk, Richard Hoshino, Ken-ichi Kawarabayashi, and Stephan Westphal. Solving the traveling tournament problem by packing three-vertex paths. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2271–2277, 2014.

[75] Ian Goodfellow, Yoshua Bengio, Aaron Courville, et al. Deep learning book. *MIT Press*, 521(7553):800, 2016.

[76] Andrea Grosso, Marco Locatelli, and Wayne Pullan. Simple ingredients leading to very efficient heuristics for the maximum clique problem. *Journal of Heuristics*, 14 (6):587–612, 2008.

[77] Georg Hager and Gerhard Wellein. *Introduction to high performance computing for scientists and engineers*. CRC Press, 2010.

[78] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[79] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[80] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[81] Mhand Hifi and Toufik Saadi. A parallel algorithm for two-staged two-dimensional fixed-orientation cutting problems. *Computational Optimization and Applications*, 51(2):783–807, 2012.

[82] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[83] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[84] Marc Huber and Günther R Raidl. Learning beam search: Utilizing machine learning to guide beam search for solving combinatorial optimization problems. In *International Conference on Machine Learning, Optimization, and Data Science*, volume 13164 of *LNCS*, pages 283–298. Springer, 2021.

[85] Lars M Hvattum, Arne Løkketangen, and Gilbert Laporte. Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. *Transportation Science*, 40(4):421–438, 2006.

[86] Soumia Ichoua, Michel Gendreau, and Jean-Yves Potvin. Exploiting knowledge about future demands for real-time vehicle dispatching. *Transportation Science*, 40 (2):211–225, 2006.

[87] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[88] Stefan Irnich. A new branch-and-price algorithm for the traveling tournament problem. *European Journal of Operational Research*, 204(2):218–228, 2010.

[89] Waldy Joe and Hoong Chuin Lau. Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 394–402, 2020.

[90] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[91] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[92] Mathias A Klapp, Alan L Erera, and Alejandro Toriello. The dynamic dispatch waves problem for same-day delivery. *European Journal of Operational Research*, 271(2):519–534, 2018.

[93] Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[94] Richard E Korf and Ariel Felner. Disjoint pattern database heuristics. *Artificial intelligence*, 134(1-2):9–22, 2002.

[95] Richard E Korf and Larry A Taylor. Finding optimal solutions to the twenty-four puzzle. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1202–1207, 1996.

[96] Richard E Korf, Weixiong Zhang, Ignacio Thayer, and Heath Hohwald. Frontier search. *Journal of the ACM*, 52(5):715–748, 2005.

[97] Slawomir Koziel, David E Ciaurri, and Leifur Leifsson. Surrogate-based methods. In *Computational optimization, methods and algorithms*, pages 33–59. Springer, 2011.

[98] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[99] Glenn Langford. An improved neighbourhood for the traveling tournament problem. *arXiv preprint arXiv:1007.0501*, 2010.

[100] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[101] Luc Libralesso, Pablo Andres Focke, Aurélien Secardin, and Vincent Jost. Iterative beam search algorithms for the permutation flowshop. *European Journal of Operational Research*, 301(1):217–234, 2022.

[102] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. *Advances in neural information processing systems*, 16, 2003.

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

[104] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. *Iterated local search*. Springer, 2003.

[105] Bruce T Lowerre. *The HARPY speech recognition system*. Carnegie Mellon University, 1976.

[106] David Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM (JACM)*, 25(2):322–336, 1978.

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

[108] Snežana Mitrović-Minić and Gilbert Laporte. Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(7):635–655, 2004.

[109] Snežana Mitrović-Minić, Ramesh Krishnamurti, and Gilbert Laporte. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(8):669–685, 2004.

[110] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[111] Andrea Mor and Maria Grazia Speranza. Vehicle routing problems over time: a survey. *Annals of Operations Research*, 314(1):255–275, 2022.

[112] Allen Newell, Herbert Alexander Simon, et al. *Human problem solving*. Prentice-hall Englewood Cliffs, NJ, 1972.

[113] Peng Si Ow and Thomas E Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26(1):35–62, 1988.

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

[115] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[116] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

[117] David Pisinger and Stefan Ropke. A general heuristic for node routing problems. *Computers & Operations Research*, 34:2403–2435, 2007.

[118] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4):193–204, 1970.

[119] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[120] Warren B Powell. A unified framework for stochastic optimization. *European Journal of Operational Research*, 275(3):795–821, 2019.

[121] Harilaos N Psaraftis, Min Wen, and Christos A Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31, 2016.

[122] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[123] Rasmus V Rasmussen and Michael A Trick. A benders approach for the constrained minimum break problem. *European Journal of Operational Research*, 177(1):198–213, 2007.

[124] Celso C Ribeiro and Sebastián Urrutia. Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research*, 179(3):775–787, 2007.

[125] Ulrike Ritzinger and Jakob Puchinger. Hybrid metaheuristics for dynamic and stochastic vehicle routing. In *Hybrid Metaheuristics*, volume 434 of *SCI*, pages 77–95. Springer, 2013.

[126] Ulrike Ritzinger, Jakob Puchinger, and Richard F Hartl. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1):215–231, 2016.

[127] 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, 2006.

[128] Roberto Maria Rosati, Matteo Petris, Luca Di Gaspero, and Andrea Schaerf. Multi-neighborhood simulated annealing for the sports timetabling competition ITC2021. *Journal of Scheduling*, 25(3):301–319, 2022.

[129] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[130] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[131] Thomas P. Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000.

[132] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach, global edition 4th. *Foundations*, 19:23, 2021.

[133] Michael Schilde, Karl F Doerner, and Richer F Hartl. Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. *Computers and Operations Research*, 38(12):1719–1730, 2011.

[134] Avik Sengupta and Alan Edelman. *Julia High Performance - Second Edition*. Packt, 2019.

[135] Paul Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. Technical report, APES Group, Dept. of Computer Science, University of Strathclyde, Glasgow, UK, 1997.

[136] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, volume 1520 of *LNCS*, pages 417–431. Springer, 1998.

[137] Benjamin C Shelbourne, Maria Battarra, and Chris N Potts. The vehicle routing problem with release and due dates. *INFORMS Journal on Computing*, 29(4): 705–723, 2017.

[138] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[139] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[140] Roni Stern, Nathan R Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Satish Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*, 2019.

[141] Nathan Sturtevant and Ariel Felner. A brief history and recent achievements in bidirectional search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

[142] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[143] Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[144] Clemens Thielen and Stephan Westphal. Complexity of the traveling tournament problem. *Theoretical Computer Science*, 412(4-5):345–351, 2011.

[145] Barrett W Thomas. Waiting strategies for anticipating service requests from known customer locations. *Transportation Science*, 41(3):319–331, 2007.

[146] Michael A Trick. Second dimacs challenge test problems. In *Cliques, Coloring, and Satisfiability*, pages 653–657, 1993.

[147] Marlin W Ulmer. Delivery deadlines in same-day delivery. *Logistics Research*, 10 (3):1–15, 2017.

[148] Marlin W Ulmer and Barrett W Thomas. Same-day delivery with heterogeneous fleets of drones and vehicles. *Networks*, 72(4):475–505, 2018.

[149] Marlin W Ulmer, Dirk C Mattfeld, and Felix Köster. Budgeting time for dynamic vehicle routing with stochastic customer requests. *Transportation Science*, 52(1): 20–37, 2018.

[150] Marlin W Ulmer, Barrett W Thomas, and Dirk C Mattfeld. Preemptive depot returns for dynamic same-day delivery. *EURO Journal on Transportation and Logistics*, 8(4):327–361, 2019.

[151] Marlin W Ulmer, Barrett W Thomas, Ann Melissa Campbell, and Nicholas Woyak. The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times. *Transportation Science*, 55(1):75–100, 2021.

[152] Sebastián Urrutia, Celso C Ribeiro, and Rafael A Melo. A new lower bound to the traveling tournament problem. In *2007 IEEE Symposium on Computational Intelligence in Scheduling*, pages 15–18. IEEE, 2007.

[153] David C Uthus, Patricia J Riddle, and Hans W Guesgen. An ant colony optimization approach to the traveling tournament problem. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 81–88. ACM, 2009.

[154] David C Uthus, Patricia J Riddle, and Hans W Guesgen. DFS* and the traveling tournament problem. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *LNCS*, pages 279–293. Springer, 2009.

[155] David C Uthus, Patricia J Riddle, and Hans W Guesgen. Solving the traveling tournament problem with iterative-deepening A*. *Journal of Scheduling*, 15(5): 601–614, 2012.

[156] Eva Vallada, Rubén Ruiz, and Jose M Framinan. New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, 240(3):666–677, 2015.

[157] David Van Bulck, Dries Goossens, Jörn Schönberger, and Mario Guajardo. Robinx: A three-field classification and unified data format for round-robin sports timetabling. *European Journal of Operational Research*, 280(2):568–580, 2020.

[158] Wouter JA Van Heeswijk, Martijn RK Mes, and Johannes MJ Schutten. The delivery dispatching problem with time windows for urban consolidation centers. *Transportation science*, 53(1):203–221, 2019.

[159] Pascal Van Hentenryck and Yannis Vergados. Traveling tournament scheduling: A systematic evaluation of simulated annealling. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, volume 3990 of *LNCS*, pages 228–243. Springer, 2006.

[160] Pascal Van Hentenryck and Yannis Vergados. Population-based simulated annealing for traveling tournaments. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 1*, AAAI'07, page 267–272. AAAI Press, 2007.

[161] Vladimir Vapnik. *The nature of statistical learning theory*. Springer, 1999.

[162] Stacy A Voccia, Ann Melissa Campbell, and Barrett W Thomas. The same-day delivery problem for online purchases. *Transportation Science*, 53(1):167–184, 2019.

[163] Stefan Vonolfen and Michael Affenzeller. Distribution of waiting time for dynamic pickup and delivery problems. *Annals of Operations Research*, 236:359–382, 2016.

[164] Christopher Wilt and Wheeler Ruml. When does weighted A* fail? *International Symposium on Combinatorial Search*, 3(1), 2012.

[165] Weixiong Zhang. Complete anytime beam search. In *AAAI/IAAI*, pages 425–430, 1998.

[166] Yang Zhang and Eric A Hansen. Parallel breadth-first heuristic search on a shared-memory architecture. In *AAAI-06 Workshop on Heuristic Search, Memory-Based Heuristics and Their Applications*, 2006.

[167] Rong Zhou and Eric A Hansen. Beam-stack search: Integrating backtracking with beam search. In *ICAPS*, pages 90–98, 2005.

[168] Rong Zhou and Eric A Hansen. Breadth-first heuristic search. *Artificial Intelligence*, 170(4-5):385–408, 2006.

Nikolaus Frohner

Project Assistant, TU Wien

+43 (720) 345 002 14
nikolaus.frohner@ac.tuwien.ac.at
www.ac.tuwien.ac.at/people/nfrohner/

Research Interests

Heuristic Optimization, Learning in Optimization, Dynamic and Stochastic Vehicle Routing, Sports League Scheduling, State-Space Search for Discrete Optimization

Activities

2022–present Reviewing for *Engineering Applications of Artificial Intelligence (EAAI)* journal

2022 Research visit for five months at the *NEO Research Group, Málaga, Spain*

2021 Research visit for five months at the *Group BONUS, Inria Nord, Lille, France*

2020–present *Annual Conference on machine Learning, Opt. and Data science (LOD)* PC member

Vienna Graduate School on Computational Optimization (VGSCO) associated PhD Student

Education

2018–present **Ph.D. studies (in progress)**, *TU Wien*, Vienna
Computer Science, supervised by Günther Raidl

2015–2017 **Master of Science (M.S.)**, *TU Wien*, Vienna
Computer Science

2008–2010 **Bachelor of Science (B.S.)**, *TU Wien*, Vienna
Computer Science

2005–2008 **Bachelor of Science (B.S.)**, *TU Wien*, Vienna
Physics

Master thesis

title *Track Finding and Filtering in the Barrel Detector of the CMS experiment using the Combinatorial Kalman Filter enhanced by Neural Networks*

supervisor Rudolf Frühwirth, Institute of High Energy Physics, Austrian Academy of Sciences, Vienna

Professional Experience

2018–present **University/Project Assistant**, *TU Wien*, Vienna
Teaching and research in discrete optimization

2014–2018 **Systems Architect**, *Infocall*, Deutsch-Wagram
Design, implementation, and operations of system architecture of emergency notification system

2010–2014 **VoIP Engineer**, *fairytel communications*, Vienna
Design, implementation, and operations of IT architecture of VoIP provider

2006–2010 **Web Developer**, *MP2 IT-Solutions*, Vienna
Project handling, technical design, and implementation of websites in health sector

Languages

German	native
English	fluent
Spanish,French	basic

Teaching

Winter Term	Lecture and Exercise *Heuristic Optimization Techniques*, Seminar on *Algorithms*, Seminar *Scientific Research and Writing*
Summer Term	Lecture and Exercise *Algorithms and Data Structures*, Seminar on *Algorithms*, Seminar *Scientific Research and Writing*

Talks

2023-03-17	Approaches to Same-Day Delivery Problems with Soft Deadlines VWCO 2023, Vienna, Austria
2022-08-31	Overview of Approaches to a Real-World Same-Hour Delivery Problem PATAT 2022, Leuven, Belgium
2022-08-29	Parallel Beam Search for Combinatorial Optimization ICPP 2022, online
2022-02-22	Learning Value Functions for Same-Day Delivery Problems in the Tardiness Regime EUROCAST 2022, Las Palmas de Gran Canaria, Spain
2020-11-23	Route Duration Prediction in a Stochastic and Dynamic Vehicle Routing Problem with Short Delivery Deadlines, ISM 2020, online
2020-09-24	*Approaching the Traveling Tournament Problem with Randomized Beam Search* CPAIOR 2020, online
2020-04-17	*A Beam Search Approach to the Traveling Tournament Problem* EvoCOP 2020, Held as Part of EvoStar 2020, online
2019-09-11	*Merging Quality Estimation for Binary Decision Diagrams with Binary Classifiers* LOD 2019, Certosa di Pontignano, Siena, Tuscany, Italy
2019-09-30	*Towards Improving Merging Heuristics for Binary Decision Diagrams* LION 13, Chania, Crete, Greece
2019-02-18	*Casual Employee Scheduling with Constraint Programming, Ant Colony Opt., and VND* EUROCAST 2019, Las Palmas de Gran Canaria, Spain

Publications

N. Frohner and G. R. Raidl, "Learning value functions for same-day delivery problems," in *Computer Aided Systems Theory – EUROCAST 2022* (R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, eds.), LNCS, Springer, 2022. accepted.

N. Frohner, B. Neumann, G. Pace, and G. R. Raidl, "Approaching the traveling tournament problem with randomized beam search," *Evolutionary Computation Journal*, 2022. in press.

N. Frohner, J. Gmys, N. Melab, G. R. Raidl, and E. Talbi, "Parallel beam search for combinatorial optimization," in *51th International Conference on Parallel Processing Workshop*, ICPP Workshops '22, Association for Computing Machinery, 2022.

N. Frohner, J. Gmys, N. Melab, G. R. Raidl, and E. Talbi, "Parallel beam search for combi-

natorial optimization (extended abstract)," in *International Symposium on Combinatorial Search*, vol. 15, pp. 273–275, 2022.

A. Bracher, N. Frohner, and G. R. Raidl, "Learning surrogate functions for the short-horizon planning in same-day delivery problems," in *17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR'21)* (P. J. Stuckey, ed.), vol. 12735 of *LNCS*, (Vienna, Austria), pp. 283–298, Springer, 2021.

N. Frohner, M. Horn, and G. R. Raidl, "Route duration prediction in a stochastic and dynamic vehicle routing problem with short delivery deadlines," *Procedia Computer Science*, vol. 180, pp. 366–370, 2021. Proceedings of the 2nd International Conference on Industry 4.0 and Smart Manufacturing (ISM 2020).

M. Horn, N. Frohner, and G. R. Raidl, "Driver shift planning for an online store with short delivery times," *Procedia Computer Science*, vol. 180, pp. 517–524, 2021. Proceedings of the 2nd International Conference on Industry 4.0 and Smart Manufacturing (ISM 2020).

N. Frohner and G. R. Raidl, "A double-horizon approach to a purely dynamic and stochastic vehicle routing problem with delivery deadlines and shift flexibility," in *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling - PATAT 2021: Volume I* (P. D. Causmaecker, E. Özcan, and G. V. Berghe, eds.), (Bruges, Belgium), 2020.

N. Frohner, B. Neumann, and G. R. Raidl, "A beam search approach to the traveling tournament problem," in *Evolutionary Computation in Combinatorial Optimization – 20th European Conference, EvoCOP 2020, Held as Part of EvoStar 2020* (L. Paquete and C. Zarges, eds.), vol. 12102 of *LNCS*, (Sevilla, Spain), pp. 67–82, Springer, 2020.

N. Frohner, S. Teuschl, and G. R. Raidl, "Casual employee scheduling with constraint programming and metaheuristics," in *Computer Aided Systems Theory – EUROCAST 2019* (R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, eds.), vol. 12013 of *LNCS*, (Gran Canaria, Spain), pp. 279–287, Springer, 2020.

N. Frohner and G. R. Raidl, "Merging quality estimation for binary decision diagrams with binary classifiers," in *Machine Learning, Optimization, and Data Science – 5th International Conference, LOD 2019* (G. Nicosia, P. Pardalos, R. Umeton, G. Giuffrida, and V. Sciacca, eds.), vol. 11943 of *LNCS*, (Siena, Italy), pp. 445–457, Springer, 2019.

N. Frohner and G. R. Raidl, "Towards improving merging heuristics for binary decision diagrams," in *Proceedings of LION 13 – 13th International Conference on Learning and Intelligent Optimization*, vol. 11968 of *LNCS*, pp. 30–45, Springer, 2019.