TU WIEN Informatics

Randomized Construction Approaches to the Traveling Tournament Problem using Lower Bound Based Heuristics

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Giulio Pace, BSc
Matrikelnummer 11835706

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Univ.-Ass. Dipl.-Ing. Nikolaus Frohner

Wien, 11. März 2021

Giulio Pace	Günther Raidl

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

TU WIEN Informatics

Randomized Construction Approaches to the Traveling Tournament Problem using Lower Bound Based Heuristics

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Giulio Pace, BSc
Registration Number 11835706

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Assistance: Univ.-Ass. Dipl.-Ing. Nikolaus Frohner

Vienna, 11th March, 2021

Giulio Pace	Günther Raidl

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

Erklärung zur Verfassung der Arbeit

Giulio Pace, 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, 11. März 2021

Giulio Pace

Acknowledgements

I would like to thank my supervisor Günther Raidl for giving me the opportunity to write this thesis and providing advice and feedback. Likewise, I want to express my deepest gratitude to Nikolaus Frohner for professional guidance and extreme availability at any time of day and night.

This work could not have been accomplished without the help of the very special people I am lucky enough to have in my life.

First and foremost Shiori, for being my therapy cat and pushing me forwards in the hardest moments of this process.

Matthias, for being my go-to-guy for every possible doubt I could have, and for helping me to dip my toes in the intricacies of the German language.

Marco, for providing interesting out-of-the-box ideas and being my personal human debugging rubber duck.

Ivan for providing mathematical guidance in times of need.

All of my old friends from Italy and the new ones from Vienna, you truly made the hard times look less hard.

Infine, il ringraziamento più grande va alla mia famiglia, che ha sempre creduto in me e mi ha sostenuto in ogni momento.

Kurzfassung

Das Traveling Tournament Problem (TTP) ist ein $\mathcal{NP}$-schweres kombinatorisches Optimierungsproblem, welches für seine praktische Schwierigkeit bekannt ist. Das Ziel ist es, einen Ligaspielplan für eine Doppelrundenturnier bei einer vorgegebene Anzahl an Teams zu entwerfen, wobei die von allen in Summe zurückgelegte Wegstrecke so gering wie möglich sein soll. Jedes Team beginnt und beendet die Saison an ihrem Heimspielort und reist bei zwei aufeinander folgenden Auswärtsspielen direkt vom ersten Spielort zum zweiten. Die Anzahl der aufeinanderfolgenden Heim- und Auswärtsspiele, die ein Team bestreiten darf, ist begrenzt. Zudem können zwei Gegner nicht in aufeinanderfolgenden Runden gegeneinander antreten. Diese Einschränkungen erschweren es zusätzlich, eine geeignete Lösung zu finden. Derzeit können von den Standard-Benchmark-Sets nur jene mit bis zu 10 Teams optimal gelöst werden, solche mit 12 Teams nicht. In dieser Arbeit werden konstruktive Lösungsalgorithmen für das Problem untersucht und mit independent lower bound (ILB) basierten Heuristiken erweitert, welche als Tourenplanungsproblem mit beschränkter Kapazität formuliert sind. Aufbauend auf einer Zustandsraum-Repräsentation für das TTP stellen wir einen $\mathcal{MAX}$-$\mathcal{MIN}$ Ameisenalgorithmen vor, welcher auf einer erfolgreichen Anwendung aus der Literatur basiert. Darüber hinaus erweitern wir einen randomisierten Beam Search Ansatz, um Instanzen mit bis zu 26 Teams bewältigen zu können, indem wir die Guidance-Heuristik näherungsweise mit Google OR-Tools berechnen. Die Auswirkungen zahlreicher Parameter des Algorithmus (etwa die Verwendung einer eingeschränkten Kandidatenliste oder eine unterschiedliche Balkengröße bei Beam Search) bei der Lösung der Tourenplanungsprobleme und der TTP werden systematisch untersucht. Bei sorgfältig eingestellten Anfangsparametern werden diese durch automatisiertes Parameter-Tuning weiter verbessert und validiert. Die Algorithmen werden anhand der NL-, NFL-, galaxy-, CIRC- und Super-Instanzen mit 10 bis 26 Teams evaluiert. Unser ACO-Ansatz weist eine Differenz in der relativen Optimalitätslücke von ca. 15-20% zu den besten bekannten Lösungen auf, was zu dem Schluss führt, dass eine Hybridlösung mit einer auf lokaler Suche basierten Methode notwendig ist, um konkurrenzfähig zu sein. Bei unserem Beam Search Ansatz ist der Effekt der Guidance-Heuristik größer: hier sind die relativen Optimalitätslücke zu den besten bekannten Lösungen kleiner als 6%; im Durchschnitt betragen sie nur 2,7%.

Abstract

The traveling tournament problem (TTP) is an $\mathcal{NP}$-hard combinatorial optimization problem famous for its practical hardness. The goal is to construct a double round robin sport league schedule for a certain number of teams, minimizing the sum of the total traveled distance over all teams. Each team starts and ends the season at their home venue and, when playing two away opponents in a row, will travel directly from the first venue to the second. There is a limit on the number of consecutive home and away games that a team is allowed to play and two opponents cannot face each other in two consecutive rounds. As of today, from the standard benchmark sets only instances up to ten teams have been solved to optimality, those with 12 have not.

The goal of this work is to study the behavior and augment constructive approaches with guidance by independent lower bound based heuristics formulated as capacitated vehicle routing problems (CVRPs). Building upon a state-space formulation for the TTP, we propose a $\mathcal{MAX}$-$\mathcal{MIN}$ ant system algorithm, based on a successful application in the literature. Furthermore, we extend a randomized beam search approach to be able to tackle instances up to 26 teams by calculating the guidance heuristics approximately using Google OR-Tools.

The effects of numerous algorithmic parameters (e.g., using a restricted candidate list during construction, varying the beam width for beam search) when solving the CVRPs and the TTP itself are presented in a detailed computational study. Sensible choices of parameters are performed by manual tuning and further strengthened by automated parameter tuning. A final comparison is performed on the NL, NFL, galaxy, CIRC and Super benchmark instances up to 26 teams. Our ant colony optimization approach generally presents relative gap differences to the best known solutions of around 15-20%, which leads us to conclude that the hybridization with a local search based method is necessary to reach competitive results. The beam search approach profits much more from the heuristic guidance and shows relative gaps to the best found solution not higher than 6% and in the mean of only 2.7%.

Contents

CHAPTER 1

Introduction

The Traveling Tournament Problem (TTP) is an optimization problem introduced in 2001 by Easton, Nemhauser and Trick [ENT01]. It emerges from the hardness to create schedules for big tournaments such as the National Football League (NFL) or the Major League Baseball (MLB). In the TTP, a league consists of a double round robin schedule where every team needs to play against every other team twice, once at their own home venue and once at their opponent's venue. The goal of the problem is to minimize the total distance over all teams during the season. There are two additional constraints to the problem, namely the *no-repeat* and *at-most* constraints. The *no-repeat* constraint prevents two teams from playing each other in consecutive rounds, while the *at-most* ensures that any given team does not play more than three times consecutively at home or away. Since its first formulation in 2001, the problem proved to be extremely challenging: at the time of writing, there are no proven optimal solutions for leagues with more than 10 teams for most of the common benchmark instances in the literature, except for the special case where all teams are the unit distance apart. Those instance and their latest solutions can be found on Michael Trick's website [1].

The goal of this thesis is to analyze the behavior and improve constructive approaches with guidance by independent lower bound based heuristics formulated as capacitated vehicle routing problems (CVRP). For this reason we propose a $\mathcal{MAX}$-$\mathcal{MIN}$ ant system algorithm, based on the approach adopted by Uthus et al. [URG09a]. Moreover, we extend the randomized beam search approach presented by [FNR20] by calculating the guidance heuristics approximately using Google OR-Tools, so that it can tackle instances up to 26 teams.

We test our approaches to understand the impact that different parameters have on the results yielded when solving the CVRPs and the TTP itself, and we present our findings in a detailed computational study. Sensible choices of parameters are performed

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

by manual tuning and further strengthened by automated parameter tuning using irace [LIDLP+16].Our ant colony optimization approach generally presents relative gap differences to the best known solutions of around 15-20%, which leads us to believe that in order to reach competitive results it is necessary an hybridization with a local search based method. The beam search approach profits much more from the heuristic guidance and shows relative gaps to the best found solution not higher than 6% and in the mean of 2.5%.

1.1 Aim of the Work

In 2019, Frohner, Neumann and Raidl published a paper [FNR20] where they presented a beam search approach to the problem. The results look very promising and they are worth further exploration. In particular, it would be interesting to find different methods to calculate a heuristic estimate for partial solutions. Since the current method is based on precalculating the lower bounds, it struggles to deal with larger instances for memory and runtime reasons due to the combinatorial explosion of the underlying problem. To tackle instances with more than 18 teams, it could be reasonable to use a heuristic value calculated on the fly, to see whether randomized constructive approaches on the same state space could be as successful as the previous method. In the paper by Frohner, Neumann and Raidl, it is clear that lower bounds are a major factor for evaluating the quality of partial solutions during the construction of solutions. The current idea to obtain lower bounds stems from the one introduced by Easton et al. [ENT01], the independent lower bounds (ILB), which was later refined by Uthus et al. [URG09b] for partial solutions. More specifically, the problem is approximated by a series of Capacitated Vehicle Routing Problem (CVRP) (see [URM07]), one for each team. The sum of the optimal solution values for each team will be a lower bound to the optimal feasibility completion in a corresponding state of the given TTP problem instance. This is a reasonable model because by setting up the demand of each city to 1 and limiting the capacity to a maximum of 3, it is easy to enforce the *at-most* constraint; the bound still neglects the *no-repeat* constraint.

In the paper [FNR20], the lower bounds are precalculated for all the states that can occur in the given instance. This approach works very well for smaller problems but quickly becomes unsustainable when the number of teams is higher than 18–20 due to the huge computation time required. We therefore want to calculate the lower bounds only when it is needed. This process can be done using Google OR-Tools [PF] by solving the vehicle routing problems heuristically, and in this work we want to ensure that the results are of a comparable quality and are obtained in a reasonable time. OR-Tools is an open source suite for optimization developed by Google, that contains several state of the art solvers tuned for some of the more common optimization problems such as integer and linear programming, constraint programming, vehicle routing and flows.

The second part of the thesis focuses on developing a randomized construction algorithm for the TTP that makes use of the heuristic of the first part. This algorithm will also have

modular improvements such as Backtracking, Backjumping and Restricted Candidate List to quickly create feasible solution of already reasonable quality. Moreover, there will be several methods to calculate the heuristic values. This should be subsequently integrated into an Ant Colony Optimization (ACO) algorithm, to improve the quality of the found solutions over time by reinforcement learning. This would not be the first time someone attempts to use an ACO algorithm for TTP, with Uthus, Riddle and Guesgen [URG09a] being one notable example. What would differentiate our approach from the previous paper would be to successfully combine the use of pheromones with the heuristic information (provided either by lower bound precalculation or Google OR-Tools). Uthus, Riddle and Guesgen [URG09a] also incorporated heuristic information (taking t he increase of the travel distance for playing a game), but they had more success without any

1.2 Methodological Approach

The methodological approach is as follows:

- At first a literature research will be conducted. The goal is to familiarize with the problem and get up to speed with the state of the art for both the TTP problem in general and the tools that will need to be used;
- We will formally model TTP, as well as a version of CVRP that applies to partial solutions to be used as a lower bound for beam search.
- An independent implementation of the CVRP calculation with Google OR-Tools will be written, in order to be able to test the running times and the quality of the lower bounds in comparison to the already existing ones. A computational study will then be conducted to verify the feasibility of this approach;
- The implementation of CVRP will be integrated to the existing version of beam search to provide lower bound guidance.
- We will then test how CVRP performs in beam search and compare it with existing results from the literature.
- The ACO Algorithm will be modeled theoretically. During this section, we will decide what specific variants of ACO could be interesting to explore and what modular improvements we want to implement;
- Afterwards, an iterated randomized construction algorithm will be implemented; after a first simple version, different heuristic methods and other modules will be implemented;
- A first computational study will be conducted, to evaluate the quality of the different heuristic methods and compare the effectiveness of the modules of running time and solution quality. Furthermore, a first parameter tuning will be performed;

- The pheromones will be added in respect to the model previously designed;
- Using irace [LIDLP+16], the algorithmic parameters will be tuned to optimize its performance;
- Another computational study will be performed to find the best performing setup and compare the results to relevant papers.

1.3 Structure of the Work

In Chapter 2 we will present an overview of the literature for the main topics of the thesis. Specifically we will discuss the state of the art approaches to tackle the TTP and CVRP. Chapter 3 will give an overview on the methodologies used to create the solution approaches for this thesis such as Heuristic Optimization, ACO and Lower Bound Based Heuristics. Chapter 4 will formally define the TTP problem, including a space formulation to represent the solution space. In Chapter 5 we will describe in detail how we calculate guiding heuristics based on the independent lower bound for the TTP, how we develop the randomized construction algorithm and the ant colony optimization approach. In Chapter 6 an overview of the computational results will be given. Finally, in Chapter 7 we will critically analyze our approach, sum up the findings of the thesis and discuss possible future work.

CHAPTER 2

State of the Art

In this chapter we will discuss the state of the art of the traveling tournament problem and the capacitated vehicle routing problem, providing an explanation for both problems as well as a summary of the more successful and meaningful attempts to solve them.

2.1 Traveling tournament problem

In this section we will discuss the traveling tournament problem, its complexity and some solution approaches that yielded interesting results.

2.1.1 Overview

The Traveling Tournament Problem was originally formulated in 2001 by Easton, Nemhauser and Trick [ENT01]. A double round tournament is a set of games where an even number of teams n plays against each and every other of them exactly twice. Of the two matches that a team plays against each opponent, exactly one of them is played at their home venue and one at their opponent's home venue. A game is defined as an ordered couple of different teams. The schedule for a double round robin of n teams requires exactly $2n - 2$ rounds, with $\frac{n}{2}$ games in each round. If a team plays multiple away games in a row, it travels from the venue of the first opponent to the venue of the second without having to go back to its own home venue; we call this a road trip. If instead it plays multiple home games in a row it just does not travel any distance, and that is called a home stand. The length of a road trip or a home stand is not measured in actual distance but in the number of opponents played. At the start and at the end of the tournament, each team has to be at their own home venue. Therefore the teams that play their last game of the season away need to get back as an additional trip. To store the respective distances between the teams, we use an n by n distance matrix named D.

We can now introduce the formal definition of TTP introduced by Easton et al. [ENT01]:

Definition: *The TTP is defined as follows:*

***Input**: n, the number of teams; D an n by n integer distance matrix; L, U integer parameters.*

***Output**: A double round robin tournament on the n teams such that*

- *The length of every home stand and road trip is between L and U inclusive, and*
- *The total distance traveled by the teams is minimized.*

When the value of U is small, teams will be forced to return home more often and therefore the total traveled distance will be higher. In contrast, if we assign $U = n - 1$ a team could potentially take a road trip where it visits all the other teams before going back home; this trip would be equivalent to a traveling salesperson tour. In this thesis we only consider $U = 3$, as is common in the literature. Furthermore, Thielen and Westphal [TW11] proved that the corresponding decision variant of the problem is strongly NP-complete by a reduction from 3-satisfiability (3-SAT). This result follows the finding presented in [Bha09], where it was proven that the TTP variant where the *at-most* constraint is not considered is NP-hard.

2.1.2 Literature

After the paper from Easton et al. [ENT01], several authors tried to tackle the problem in different ways. In this section, we will give an overview of some of the most significant ones.

Easton et al. [ENT02] propose a branch-and-price algorithm that manages to find the first provable optimal solutions for an instance of size 8. Their approach combines integer programming for the master problem and constraint programming to solve the pricing problem and as a primal heuristic.

Stefan Irnich [Irn10] presents another branch-and-price approach that manages find proven optimal solutions for some instances such as NL8 by using a restricted shortest path problem as pricing problem.

In 2006, Anagnostopoulos et al. [AMVHV06], they present a traditional simulated annealing approach. The algorithm explores both feasible and infeasible schedules using a large neighborhood generated by complex moves. A reheating is performed to balance the exploration of the neighborhood and escape local minima when the temperature gets too low. This approach seems to be especially robust since the worst solutions are still quite decent. The following year Van Hentenryck and Vergados propose in [VHV07] a population-based simulated annealing algorithm, that runs in parallel and is organized in a series of waves. After each wave a macro-intensification is performed, by restarting a majority of the runs from the best found solution. Furthermore, there are some *elite waves* that survive opportunistically to obtain diversification. This method is the one that, so far, provides the best solutions for the larger instances of NL and CIRC.

Ribeiro and Urrutia [RU07] tackle the mirrored variant of TTP with a combination of GRASP and iterated local search metaheuristics that makes use of a strong neighborhood based on ejection chains. Mirrored TTP is a variation of TTP where every team plays every other team exactly once in the first $n-1$ rounds forming the first half of the schedule, followed by the same schedule with reversed venues. Some of the solutions found are, especially for large instances of size 24, better than the ones found by other approximate algorithms for regular TTP at the time of writing. This is especially impressive considering that mirrored TTP is a significant restriction. Furthermore, they present a fast and very effective construction algorithm.

Di Gaspero and Schaerf [DGS07] propose a competitive Tabu Search algorithm that uses a complex combination of neighborhood structures.

Uthus, Riddle and Guesgen [URG09b] present an hybrid between a depth-first branch-and-bound and an exact iterative deepening A* approach (IDA*). Later on they also propose [URG12] several techniques to improve IDA* like disjoint pattern databases, symmetry breaking, subtree skipping, forced deepening and elite paths. These methods managed to solve instances up to ten teams to proven optimality.

Uthus, Riddle and Guesgen [URG09a] make use of a MAX-MIN Ant System (MMAS) algorithm with ACS selection rule to drastically improve the quality of ACO algorithms for TTP. The novelty of this approach is in the hybridization with forward checking, pattern matching and conflict-directed backjumping, but the basis of it is a traditional MMAS as presented in [SH00]. They also use the ACS rule as presented in [DG97]. Another notable example of usage of an ACO was presented by Chenet al. [CKB07] and uses a framework to solve the TTP that makes use of an Ant Algorithm as a hyper-heuristic.

Miyashiro et al. [MMI12] construct a randomized approximation algorithm that makes use of a new lower bound that, for instances with n teams, provides feasible solutions with approximation ratio smaller than $2 + (9/4)/(n-1)$. This is the first approximation algorithm where the approximation ration is constant and smaller than $2 + 3/4$.

Goerigk et al. [GHKW14] publish a graph-theoretic approach to TTP, where they generate a new "canonical" schedule in which, for every team, their road trips of size three match up with the graph's minimum-weight P_3-packing. This schedule is used as input for an hybrid algorithm [GW16] that combines local search and integer programming, finding best solutions for five benchmark instances. The algorithm makes use of a powerful commercial integer programming solver.

Finally, in Frohner, Neumann and Raidl [FNR20], a Beam Search approach is presented. In order to evaluate the quality of the partial solutions, they make use of the lower bound derived from a state as introduced by Uthus et al. [URG09b, URG12]. This paper finds new best feasible solution for two benchmark instances and will be crucial in the development of this thesis.

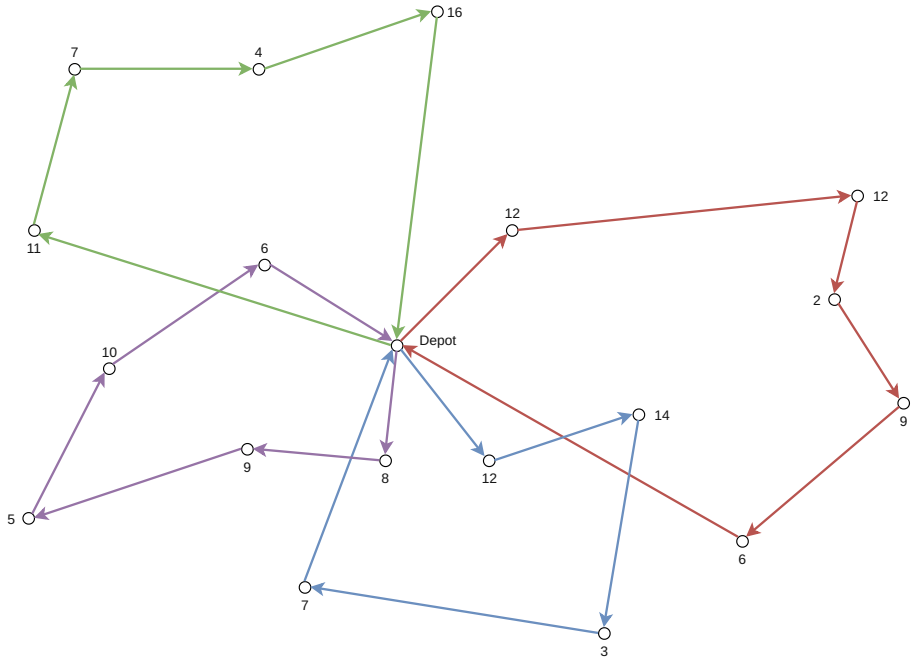

Figure 2.1: Example of a CVRP instance. For this example we consider the euclidean distances between points. Four vehicles with capacity 42 fulfill the demands of several clients. Each color represents the path of a vehicle. The green and the purple vehicle satisfy 38 requests each, blue 36 and orange 41. After visiting their clients, they go back to the depot.

2.2 Capacitated Veichle Routing Problem

The Capacitated Veichle Routing Problem (CVRP) is a popular problem in the field of heuristic optimization. In this section we will give a small overview of the problem, showing some of its variants and discussing its complexity. We will then briefly present a review on the more significant solution approaches in the literature.

2.2.1 Overview

We will mainly follow the formulation that Toth and Vigo present in the first chapter of "An Overview of Vehicle Routing Problems" [TV02]. The Vehicle Routing Problem (VRP), also known as Vehicle Scheduling Problem, is a combinatiorial optimization problem with the goal of finding the optimal set of *routes* taken by a fleet of *vehicles*, initially located in *depots*, to deliver goods to final users (*customers*) and then return to their starting depots. The vehicles perform their movements through a *road network*. The road network is generally represented by a directed graph where vertices are depot, customers and road junctions and arcs are road sections. Each arc has a cost associated to it that represent its length. To each customer we assign a *demand*, i.e. the amount of

good requested. If it is not possible to meet the demand of every customer, there could be a system of priority and penalities for each customer. The vehicles have a specific depot called *home depot*, where it must start and end its route. Every vehicle also has a cost associated to its use.

Several objectives can be considered for VRP. Some examples include: minimizing the total cost of vehicles, minimizing the number of vehicle used, minimize the penalties associated to the lack of service to some clients in cases where it is impossible to satisfy the demand of everyone. It is also possible to combine different objectives. VRP is trivially NP-hard because it includes TSP as a special case [AFGG11]. This is also the case for the CVRP variant when the capacity of the vehicles is greater than 2, as proven in [AMS05].

The Capacitated Vehicle Routing Problem (CVRP) is a variant of VRP where the demands of the clients are deterministic, known in advance and cannot be split. Furthermore, vehicles have a maximum capacity and are all identical and based in a single central depot. The objective is to minimize the total cost needed to satisfy the demands of all the customers.

The k-Customer Vehicle Routing Problem (kVRP) restricts the maximum number of customers that can be visited by each vehicle before going back to its depot. Hassin and Rubinstein [HR05] prove that UNWEIGHTED -kVRP is NP-hard for $k \geq 3$ by reducing it from a (k-1) size instance of the Partition into Paths of Length k (kPP) and that DIRECTED UNWEIGHTED -kVRP is NP-complete for $k \geq 3$ by using a reduction from 3-Dimensional Matching. From this, they derive the corollary that DIRECTED -kVRP is not 2 p(n) approximable for any $k \geq 3$ and a polynomial p, unless P = NP. Furthermore, they prove that there is a polynomial 4-approximation algorithm for 3VRP, i.e. there exists a polynomial algorithm that can calculate a solution of 3VRP within a factor of 4 times the optimal solution length.

2.2.2 Literature

This literature review will mainly follow the fifth and sixth chapter from "An Overview of Vehicle Routing Problems" by Toth and Vigo [TV02]. There is a history of exact approaches for CVRP that were successful, like the branch-and-cut algorithm proposed by Augerat et al. [ANB+95], that is especially strong for large instances, or the branch-and-price algorithm that Christiansen and Lysgaard [CL07] apply to the variant of CVRP with stochastic demands. These methods are extremely good for smaller instances, but quickly become unfeasible as the size of the instances grows, forcing us to use heuristics. Since the problem is extremely popular, during the years several approaches have been proposed. They could be grouped in two families, namely *classical heuristics* and *metaheuristics*. The methods that belong to the first family are, generally speaking, quite modest in the size of the search space, but typically guarantee decent solutions in limited computing time. Classical Heuristic are also relatively easy to generalize or adapt between different variants of CVRP. Metaheuristics, on the other hand, are much more dependent on the

specific context and on the fine tuning of the parameters. Furthermore, they require significantly more computational power. In exchange, the quality of the solutions is much higher.

The more straight forward heuristic methods are the so-called constructive methods and attempt to build a feasible solution step by step while keeping the solution cost in check. Clarke and Wright [CW64] is a very well known heuristic for the VRP that introduced the concept of *savings*; when we can merge two routes $(0, ..., i, 0)$ and $(0, j, ..., 0)$ into a single feasible one $(0, ..., i, j, ..., 0)$, we generate a saving $s_{ij} = c_{i0} + c_{0j} - c_{ij}$. These merges can be applied until there are feasible ones. Another very popular approach to constructive heuristics is the sequential insertion method that consists on starting with empty routes and then expand them one node at the time. A notable example can be found in a paper by Mole and Jameson [MJ76], where they start with a partial route, calculate the cost of inserting all the vertices that are not in the route yet and insert the best one according to parameters μ and λ. These two parameters will modify the criterion of choice for the best candidate, i.e. minimum extra distance, smallest sum of distances between two neighbours, vertex furthest from the depot. Another example of constructive heuristic was presented by Christofides et al. [Chr79]. Their insertion heuristic starts from the same two user-controlled parameter algorithm as Mole and Jameson [MJ76] but is developed in two elaborate phases that guarantee a better coverage of the search space. This method requires half the computing time and yields results that are consistently better.

Several metaheuristic methods have been applied to CVRP. Osman [Osm93] presented a very successful Simulated Annealing (SA) algorithm that can operate on an unspecified number of vehicles. Toth and Vigo presented a Granular Tabu Search (GTS) algorithm [TV03] that yielded excellent results for CVRP with maximum route length constraints. The idea behind it comes from the observation that is very rare that a long edge of the graph is part of an optimal solution. Eliminating the edges that are bigger than a *granularity threshold*, we can easily filter out several unpromising solutions. Van Breedam [VB96] implemented a Genetic Algorithm (GA) for CVRP, but mostly to compare the impact of different parameters of GA in terms of efficiency of search. In [LLLY10], an Enhanced Ant Colony Optimization (EACO) is presented; the EACO uses of SA to provide ACO a good starting solution. This approach performs better than SA and ACO on their own. Finally, [HT19] proposed a Neural Network (NN) approach that integrates a deep neural network to provide a learned heuristic to apply in a large neighborhood search (LNS) framework. The LNS applies a destroy and repair paradigm to feasible solution where the deep neural network, trained via policy gradient reinforcement learning, is in charge of the repairing. This method performs better than the best machine learning approaches existing at the time and gets close to the performance of state of the art methods.

Google released Google OR-Tools [PF], an open source software suite that offers solvers for a wide variety of optimization problems such as Scheduling, Bin Packing and several variants of VRP (including CVRP) and provides different metaheuristics, e.g. Greedy

Descent, Guided Local Search, Simulated Annealing and Tabu Search. In Chapter 5 we will explain the relevant aspects of Google OR-Tools in detail.

CHAPTER 3

Methodology

In this chapter we define and explain the concepts that will be used to design and implement our algorithms. In particular, we will define an optimization problem, introduce the concept of heuristic optimization and explain the ideas that originated beam search and ant colony algorithms. For this introductory section we will follow Papadimitriou and Steiglitz [PS98].

The Traveling Tournament Problem is a combinatorial optimization problem. A problem in the field of computational complexity could be a satisfaction problem or an optimization problem. In the former we are interested in answering a question with a boolean answer, such as "is it possible to create a path that connects a set of cities?" or "Is there a path that connect a set of cities that is shorter than a value k?". In the latter the question would be "what is the shortest path that connects a set of cities?". This introduces the concept of quality of a solution: just finding a generic solution is not enough, but we are interested in the best one. The quality of a solution is measured by an objective function, which assigns each solution a value, its objective value. We call a problem a minimization problem when the goal is to find a solution with the smallest objective value, while a maximization problem aims to find a solution with the largest. A maximization problem can be trivially converted to a minimization problem by multiplying the objective value by -1. Therefore, for consistency reasons, we will treat every problem as minimization problems in this thesis. The set of solutions for a problem could be continuous or discrete. In the latter case we call it a combinatorial optimization problem, in contrast with continuous optimization problems where we have uncountably many solutions. Optimization is, in other words, finding the minimal (or maximal) solution for a given problem from the set of all the possible solutions according to a specific metric. Therefore, we formally define a combinatorial optimization problem as follows:

Definition 3.1 (Combinatorial Optimization Problem) *A combinatorial optimiza-*

tion problem is a set of instances. An instance is a pair (S, f), where S is the finite set of all feasible solutions (also known as a solution space or search space) and f is a cost function $f: S \to \mathbb{R}$, often called objective function. The problem is to find an $x \in S$ so that $f(x) \leq f(y)\, \forall y \in S$, which is called a globally optimal solution.

3.1 Heuristic Optimization

Naively, in order to find the optimal solution for a given combinatorial optimization problem instance, one could to enumerate all the possible solutions and select the one with the one with the smaller objective. This is possible since the solution space is by definition finite. But when the search space of a problem is especially big, this approach has the obvious drawback of an extremely big cost in terms of computational time. Most popular optimization problems nowadays are NP-hard. Therefore, since just enumerating all the possible solutions would be an impossible task for any machine, there needs to be a quicker and more feasible technique, that allow to solve problems to optimality thanks to speed-ups or pruning of the search space. For example, branch and bound algorithms [LD10] use upper and lower bounds to choose which branch of the search space to explore: if a branch b of a minimization problem has a lower bound that is greater than the upper bound of a branch b', there is no reason to look into b because following b' will always yield a better solution. Branch and bound as a paradigm is the bases for a plethora of other very popular and successful algorithms like branch-and-cut [PR91] and branch-and-price [BJN$^+$98]. Another exact algorithm that is especially popular for graph traversal and path searching is A* [HNR68]. This method was introduced in 1968 it is based on selecting the path that minimizes $f(s) = g(s) + h(s)$, where $g(s)$ is the cost of the path from the start to the node s and $h(s)$ is a heuristic function that estimates the cost of the cheapest path from s to the goal. A* also introduced the concept of admissibility, a necessary property of $h(s)$. We define $h^*(s)$ as the optimal cost of reaching a goal (complete feasible solution) from state s: a heuristic $h(s)$ is admissible if and only if $h(s) \leq h^*(s)$ for every state s. In other words, a heuristic is admissible if it never overestimates the cost of optimally completing the current partial solution [RN02]. A* is a very popular and still widely used, for example in [URG12].

The aforementioned exact methods can solve surprisingly big instances to optimality, but still have exponential runtimes. To more reliably be able to solve bigger instances we need to make a compromise and give up optimality, and pursue methods that can still find good non optimal solution in polynomial time. In order to explain some possible approaches to this issue, we will from now on use the very popular and well known Minimum Hamiltonian Cycle Problem (MHCP) as a running example [BH83]. Let $G = (V, E)$ be a directed weighted graph with V vertices that represent a set of cities and E edges that represent that two cities are connected. The weights of the edges represent the distances between them. G is not necessarily complete. The goal of MHCP is to find the smallest cycle for every vertex $v \in V$. A cycle is a set of nodes where the first and the last are the same and there are no other duplicates. In other words, we are interested in finding the shortest tour that visits every city exactly once and gets

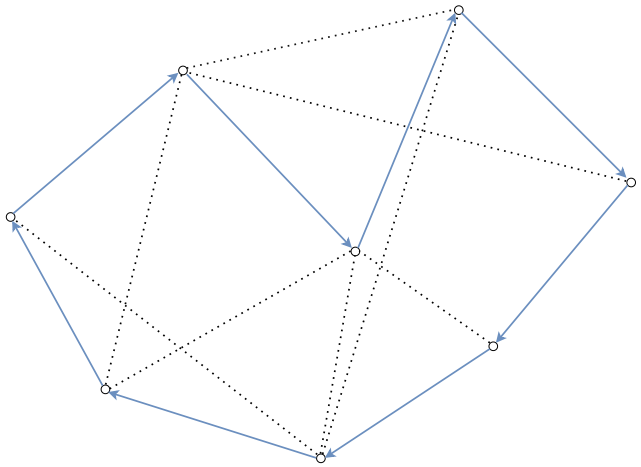

Figure 3.1: Example of an instance of the Minimal Hamiltonian Cycle Problem. The weights of the edges are the euclidean distances between the two vertices.

back to the starting city. Figure 3.1 shows an example of an instance of MHCP. The Traveling Salesperson Problem is a special case of MHCP in which G is complete. The number of possible permutations is $(n)!$, that can be reduced to $(n-1)!$ by fixing the starting point to break the rotational symmetry. Furthermore, the inversion symmetry can be broken as well, resulting into $(n-1)!/2$ equivalence classes. If the graph is not complete some of these permutations will not produce valid solutions, for example when two cities that are not connected are next to each other. In short, this means that with a sufficiently big value of n it is impossible to enumerate these solutions. The approach of Heuristic Optimization is to find a good solution, even if it is not the global best, within reasonable time. We now discuss two well-know paradigms from heuristic optimization: *construction* and *improvement*.

Construction In the construction paradigm, we start with an empty solution and add new elements to it until a complete solution is reached. This possible for problems, where the solutions can be decomposed into elements in a meaningful way. When the solution is not complete yet, it is called *partial solution*. A partial solution is stored in a *state*, that keeps all of the information necessary to continue to construct the solution. An example of this approach would be the *nearest neighbor heuristic*, a greedy algorithm in which the next city chosen when construction a tour is the closes one. Construction methods cannot guarantee neither the feasibility nor the optimality of the solution. Techniques like backtracking and backjumping [Dec90] are ways of guaranteeing feasibility. With backtracking, when reaching an unfeasible solution, the last constructive step is reverted and blacklisted. In backjumping, a more steps are reverted at the time. In Figure 3.2 we can see that the last city visited during a greedy construction with the nearest neighbor heuristic in our MHCP problem instance could not be connected to the first and therefore the solution is not feasible. When constructing a solution, we use heuristics to evaluate

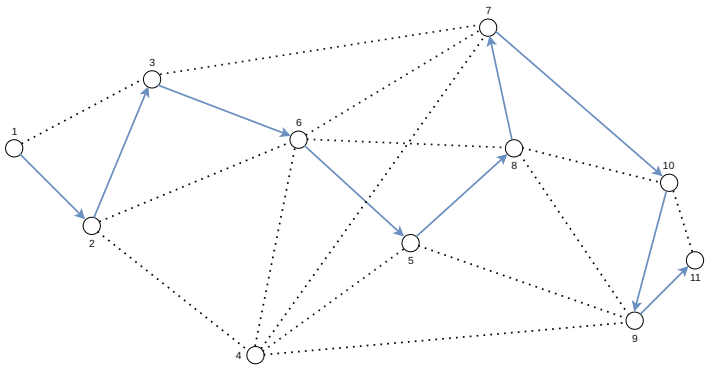

Figure 3.2: Example of a greedy construction of a MHCP instance, starting from node 1. Note how it is impossible to reach node 4 from node 11, making this solution non feasible.

extensions and to give an estimate of the cost of completing it. A heuristic is, in fact, a function that maps a state that would result when an extension would be chosen. A risk that we incur when using heuristic is that they could be myopic and only consider a "local environment" without taking the whole problem into consideration. A heuristic could be lower bound based if its value is always smaller than the actual value that it is estimating. Similarly, it could be upper bound based. An example of a reasonable lower bound based heuristic for MHCP would be the sum of the shortest incident edge on every vertex that is still to be inserted. Examples of constructive algorithms are Beam Search and Ant Colony Algorithms, that we will discuss in more detail further in the chapter.

Improvement We will now briefly discuss the improvement paradigm, where we start from a valid and sub-optimal solution and we perform local changes to improve it. We do not know if the starting solution is optimal. A *neighborhood structure* is a criterion with which we define what solutions are neighbors with the current solution. The set of neighbors $N(x)$ for a solution x is the *neighborhood* of x. To navigate the neighborhood we could perform a local search until we reach a local minimum, i.e. a solution that is better than all of its neighbors. As much as we want to reach a local minimum, we also want avoid stagnating in a weak one. For these reason techniques like tabu search [Glo86] and simulated annealing [KGV83] manage to escape local minimums to find improvement further away from our current solution. Tabu search in particular keeps track of the already visited states and blacklists them for a short period of time, increasing the chance of exploration (i.e. traversing the search space very quickly with the hope of finding promising solutions yet to be refined) over exploitation (i.e. focusing on improving a small amount of promising solutions). On a similar note, simulated annealing uses a *temperature* parameter that represents the likelihood of moving towards a state that is worse than the current one. The *temperature* will start very high, promoting exploration, and will decrease in time, to promote exploitation.

3.2 Beam Search

In this section we are going to explain the idea behind Beam Search, which is a famous constructive approach, as well as some practical applications.

3.2.1 Overview

Beam Search is a heuristic search algorithm presented for the first time in 1967 by Lowerre in [Low76]. The term "Beam Search" was coined by Ray Reddy in 1977 in [R+77]. [Sam17] defines Beam Search as a heuristic search algorithm that is the combination of breadth-first search (BFS) and best-first search. Beam search acts just like a BFS algorithm but instead of keeping every possible state in memory, it only pursues the β most promising nodes per each level of the search tree and discards the others. This gives a space requirement of $\mathcal{O}(\beta n)$ (n is the number of decisions to be made to construct a solution) but, since it is in essence still a greedy algorithm, has no guarantee of finding the optimum. A bigger value of β will in general provide a more accurate search at the price of a higher computational cost. In order to select the most promising nodes of every level, a heuristic estimation of the objective is needed. The f-value of a state s will be defined as follows:

$$f(s) = g(s) + h(s)$$

where $g(s)$ is the evaluation of the state s and $h(s)$ is a heuristic value estimating the optimal feasible completion of s. The heuristic value $h(s)$ is an estimate of the optimal cost of reaching a complete solution and its goal is to provide a guide to the search. Therefore there is a trade-off between precision, and therefore better guidance, and running time. Furthermore, we are interested in a tight heuristic value: setting every $h(s) = 0$ would be a very fast and simple, but not very precise, heuristic. Choosing the tightest heuristic helps to provide better guidance to our search method. A lower bound is defined as a value that is smaller than every element in a set [PS03]. The same idea in reverse works for upper bounds. The closer the upper and lower bounds are, the more precise our estimate will be. If the upper bound is the same value as the lower bound, in fact, that is the exact value. In Figure 3.3 we show an example of how a beam search with $\beta = 2$ solves a complete instance of MHCP of size 4. For every layer, we only check the neighbours of the β solutions that have been selected. It is interesting to see how a greedy selection method would have selected the path on the left over the one on the right based on the second selection, but eventually the one on the right is the better of the two. An example of using lower bounds as guidance could be found in [Blu05].

3.2.2 Literature

In this section we present applications of beam search from the literature in the fields of continuous speech recognition and job scheduling problems.

Ney et al. [NMNP87] present Time-Synchronous Beam Search for continuous speech recognition, a variant that processes one observation after the other and all active

$$d = \begin{pmatrix} 0 & \infty & 7 & 9 \\ \infty & 0 & 1 & 3 \\ 7 & 1 & 0 & 4 \\ 9 & 3 & 4 & 0 \end{pmatrix}$$

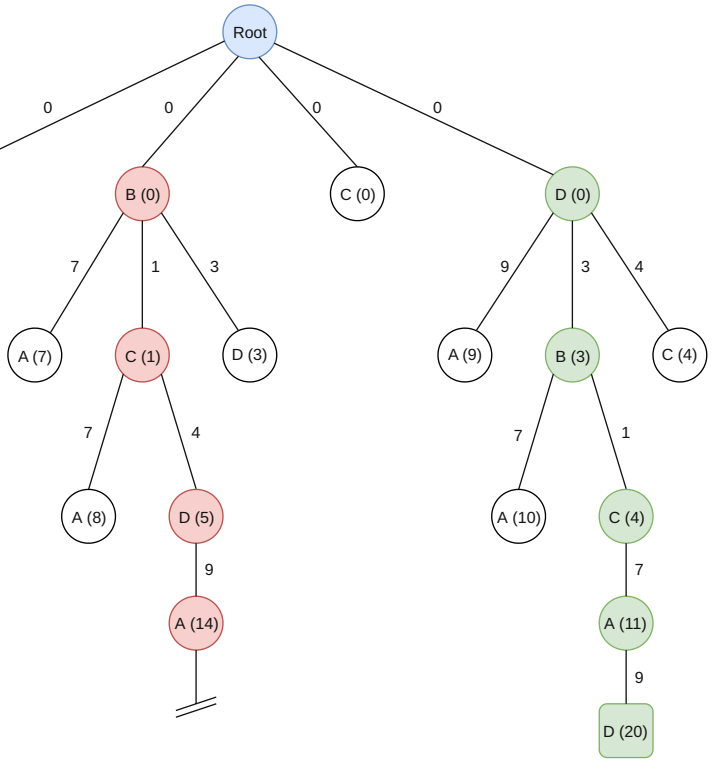

Figure 3.3: Example of a beam search of $\beta = 2$ on an instance of MHCP of size 4. Left: distance matrix of the instance. Right: representation of how beam search constructes the solutions. The f values of the nodes are in parenthesis, and the values of the edges are the cost between two nodes. For the sake of simplicity, $h(s) = 0$ and when there is a tie we select a random node between the tied ones. We start from an empty root state, and to reach the terminal state (rectangular node) we go back to the first node we selected to close the cycle. Note how the red path looks more promising than the green one at the beginning, but turns out to be infeasible.

hypotheses refer to the same point in time, in order to make comparison faster and easier. Steinbiss et al. [STN94] further improved Time-Synchronous Beam Search with two robust pruning methods, achieving a much faster average and peak search effort. The first method, called Histogram Pruning, removes areas of search space where the values of a histogram on the hypothesis score are smaller than a threshold. The second method is a *look-ahead* system for the vocabulary tree.

Ow et al. [OM88] publish a variant of Beam Search named Filtered beam search and apply it to a scheduling problem that, despite being computationally simple, produces high quality solutions. The idea behind it is that since it is rather expensive to evaluate a solution, a faster filtering method allows to apply the costly evaluation only to more promising nodes.

In [SB99], Sabuncuoglu and Bayiz apply Beam Search to the Job Shop Scheduling problem, achieving very promising results that hold up against other state of the art methods at the time of publication.

Ortmanns and Ney [ON00] show a two look-ahead technique applicable to Beam Search for large vocabulary continuous speech recognition that achieve a great improvement in performance. The two techniques are incorporated in the pruning phase of Beam Search and they are called *language model look-ahead* and *phoneme look-ahead*. With these improvements the size of the search space is reduced by 30-fold, while the computational effort is reduced by a factor of 5.

Zhou et al. [ZH05] present Beam-Stack Search, that integrates backtracking with Beam Search and guarantees to find an optimal solution. The algorithm finds a good sub-optimal solution quickly with Beam Search and then backtracks and continues to find other solutions that are better than the starting one, until eventually it converges to an optimal one.

Finally, Blum [Blu05] tackles the Open Shop Scheduling problem with an hybrid approach between Beam Search and Ant Colony Optimizaton (ACO, see Section 3.3), called Beam-ACO. This method takes advantage of the fact that ACO explores the search space in a probabilistic way, while solutions generated by BS are usually deterministic. In Beam-ACO, the beam is filled governed by the probabilistic selection rules of ACO instead of always taking the β best successors. Using an hybrid between these two methods allows a better exploration of the search space that translates in solutions that are state-of-the-art for the problem for a wide range of benchmark instances.

3.3 Ant Colony Optimization

In this section we are going to present Ant Colony Optimization and some of its variants, together with a brief summary on interesting applications. The main reference will be Chapter 8 of the book Handbook of Metaheuristic: *Ant Colony Optimization: Overview and Recent Advances*, by Marco Dorigo and Thomas Stützle [DS10]. Ant Colony Optimization (ACO) is a metaheuristic inspired by the behavior of ants in nature that is used to solve hard combinatorial optimization problems. Ants in search for food leave a trail of pheromones on their path in order to communicate with other ants. ACO algorithms use pheromone trails to guide (artificial) ants towards constructing solutions. By making probabilistic decisions based both on the trails and, if available, some heuristic information, the ants implement a randomized construction heuristic. An example could be seen in Algorithm 3.1. For this reason, ACO is commonly regarded as an improvement and an extension to construction heuristics. The crucial difference between ACO and a traditional construction heuristics lies in the fact that ACO algorithms stores the information accumulated during iterations of search in the form of pheromones in order to use it later on.

The first practical example of an ACO algorithm is Ant System (AS), proposed in

Algorithm 3.1: Ant colony optimization metaheuristic

```
Result: A feasible solution
initialization of pheromone trail ;
while termination conditions not met do
    ConstructAntSolutions ;
    ApplyLocalSearch {optional} ;
    UpdatePheromones ;
end
```

[DMC96] and applied to the Traveling Salesperson Problem. In AS, the problem is solved in t discrete timesteps. In every timestep, every ant creates a complete solution. The next element in the solution is chosen by ant k with a probability given by the following formula:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}^\beta]}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha \cdot [\eta_{il}^\beta]} & \text{if } j \in N_i^k \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Where τ_{ij} is the intensity of the pheromone trail from i to j, η_{ij} is a matrix for local information, N_i^k is the list of valid neighbors (extensions to the partial solution) and α and β are parameters that control the ratio between pheromones and local information. After timestep t, the used paths are updated for $t+1$ with the following formula:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t) \tag{3.2}$$

with

$$\Delta\tau_{ij}^k(t) = \begin{cases} 1/\text{length}(T^k(t)) & \text{if } (i,j) \in T^k(t) \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where T^k is the constructed tour for ant k. Furthermore, an evaporation will be applied to the pheromone matrix using a parameter $0 < \rho \leq 1$ as the pheromone persistance:

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) \tag{3.4}$$

More specifically, Algorithm 3.2 shows an example of pseudocode for AS on MHCP would look like the following:

The results of AS looked promising at the beginning, for example being able to find a good solution for a 75-city TSP problem [DMC96]. During the years AS has been applied to a wide variety of problems, sugh as probabilistic TSP [BBS+09], a variation of TSP in which there each city has a probability of being visited and the goal is to find an *a-priori* tour that minimizes the expected value of tours, sequential ordering [GD00],

Algorithm 3.2: Ant System algorithm

```
Output: A feasible solution S
for t ← 1, ..., t_max do
    foreach ant k = 1, ..., m do
        Choose initial city c;
        while list of unvisited cities C ≠ ∅ do
            Choose a city j from the neighborhood N_i^k of i according to p_{i,j};
            remove i from C;
            i ← j;
        end
    end
    Apply pheromones to paths;
    Evaporate pheromones;
    return S;
end
```

scheduling [CDMT94, Blu05], 2D-HP protein folding [SH05], DNA sequencing [BVB08], and packet-switched routing in Internet-like networks [DCD98].

AS is also the baseline for several variants that obtained competitive results. Stützle and Hoos [SH97] introduced MAX-MIN ant system (MMAS), a variant of AS where the pheromone values are limited by τ_{max} and τ_{min} to avoid premature stagnation. MMAS is one of the most successful ACO algorithms. Another very popular AS variation was presented by Dorigo et al. [DG97] and is called Ant Colony System (ACS). In ACS the choice of the next city j while constructing a solution is done following the following procedure:

$$j = \begin{cases} \text{argmax}_{l \in N_i^k} [\tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}] & \text{if } q \leq q_0 \\ N & \text{if } q > q_0 \end{cases} \tag{3.5}$$

where N is a city selected with the AS rule and q is a uniformly distributed random variable $0 \leq q \leq 1$. With a low value of q_0, we will have a strong diversification, because the selection will be the same as AS. On the contrary, with a high value of q_0 we will perform intensification, since we will be utilizing already gained information. Furthermore, in addition to the global pheromone update that was presented for AS, a local pheromone update was introduced, in which the pheromone is updated immediately after using a specific edge with the help of parameter ξ:

$$\tau_{ij}(t) = (1 - \xi) \cdot \tau_{ij}(t) + \xi \cdot \tau_0 \tag{3.6}$$

Update the pheromone locally means that the procedure will construct different solution if the ants build solutions in parallel or sequentially. Optionally, after the solutions have been constructed and before updating the pheromone value, one could perform a problem

specific action, commonly called daemon action [DDC99]. They generally consist in applying local search to the constructed solution, but they could also perform other problem specific centralized actions that cannot be performed by the single ants. We then use the locally improved solutions to update the pheromone.

ACO proved to be a solid metaheuristic, being able to find good results in several famous problems such as quadratic assignment [MC99, GTD97], graph coloring [CH97], vehicle routing [BHS99] and vehicle routing with time windows [GTA99]. It has also been applied to the traveling tournament problem in multiple occasions. In [CVO03], Crauwels et al. investigate how a basic ant system with local improvement techniques performs on NL instances of sizes from 4 to 16, obtaining extremely underwhelming results. The explanation that they provide is that ACO may perform well for the optimization side of the problem, but could be lackluster in the feasibility part. A more successful approach is presented by Chen et al. in [CKB07], where a framework that employs ant algorithms is used as a Hyper-heuristic for ten low level heuristics such as *swap teams* and *shift move* is able to produce good quality solution in comparison to other state-of-the-art methods. The strongest results so far were achieved by [URG09a], where ACO with the ACS selection rule is hybridized with forward checking and conflict-directed backjumping, as well as using pattern matching and other strategies for constraint satisfaction, to address the feasibility aspect of the problem. This method is able to outperform other ACO approaches at the time and produces solutions of comparable quality to results from the literature.

CHAPTER 4

Problem Formalization

In this chapter we provide formal definitions for the capacitated vehicle routing problem used to derive lower bounds for the traveling tournament problem, as well as the state space formulation that we use to represent the states of TTP while constructing a solution.

4.1 Traveling tournament problem

In this section we will provide a mathematical model for TTP as well as a State Space Formulation to represent the solution space of the TTP. Unless explicitly stated otherwise, we will adopt the notation used by Frohner et al. in [FNR20].

4.1.1 Mathematical Model

The input of our problem is a set $V = \{1, ..., n\}$ of n teams where n is even and a distance matrix d, where $d(i, j)$ is the distance that team i has to travel from its home venue to reach team j's home venue, $\forall i, j \in V$. The goal is to find a schedule that minimizes the sum of the distances traveled in a double round robin tournament by all the teams. Each team starts and ends at its home venue and is subject to the following two constraints: the *at-most* constraint, that prevents a team from playing more than U games away or at home, and the *no-repeat* constraint, that prevents teams from playing against each other in consecutive rounds.

For the remaining part of the chapter, we will adopt the formulation presented by De Werra in [DW80] and used in [RU07] and [FNR20] to represent a tournament schedule as a 1-factorization of a graph. The model consists of a complete weighted directed graph $G = (V, A)$ where each node of G represents a team and weights are the distances defined by d. As double round robing schedule is then modeled as an ordered partitioning of the arcs into $2n-2$ perfect matchings, or 1-factors. The corresponding ordered 1-factorization $T = (G^1, ..., G^{2n-2})$ with $G^r = (V, A^r)$ represents the $2n-2$ ordered rounds. An example

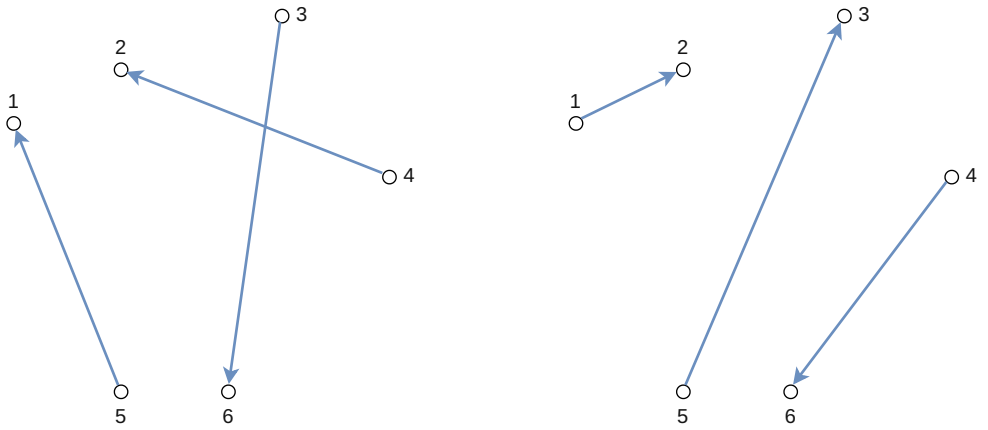

Figure 4.1: Example two 1-factors for an instance with 6 teams.

of two 1-factors can be seen in Figure 4.1. An arc between i and j in G^r means that team i is playing against team j at their venue during round r. This concept is also represented by $i \rightarrow^r j$.

We define $p_i^r \in V$ as the position of team i in round r. The values of p are determined by the arcs that are incident to i in A^r. A schedule T will then have the following objective value:

$$z(T) = \sum_{i=1}^{n} \left(d(i, p_i^1) + \sum_{r=2}^{2n-2} d(p_i^{r-1}, p_i^r) + d(p_i^{2n-2}, i) \right). \tag{4.1}$$

Therefore, let $\mathcal{T}$ be the set of all 1-factorizations of G, i.e. the set of all possible feasible round robin tournaments with n team, we define TTP with $U = 3$ as:

$$\min_{T \in \mathcal{T}} z(T), \tag{4.2}$$

subject to:

$$\begin{aligned} &((j_1 \rightarrow^r i) \wedge (j_2 \rightarrow^{r+1} i) \wedge (j_3 \rightarrow^{r+2} i)) \Rightarrow (i \rightarrow^{r+3} j_4) \\ &\forall i, j_1, j_2, j_3, j_4 = 1, ..., n \quad i \neq j_1 \neq j_2 \neq j_3 \neq j_4 \quad \forall r \in 1, ..., 2n-5 \end{aligned} \tag{4.3}$$

$$\begin{aligned} &((i \rightarrow^r j_1) \wedge (i \rightarrow^{r+1} j_2) \wedge (i \rightarrow^{r+2} j_3)) \Rightarrow (j_4 \rightarrow^{r+3} i) \\ &\forall i, j_1, j_2, j_3, j_4 = 1, ..., n \quad i \neq j_1 \neq j_2 \neq j_3 \neq j_4 \quad \forall r \in 1, ..., 2n-5 \end{aligned} \tag{4.4}$$

$$(i \rightarrow^r j) \Rightarrow \neg(j \rightarrow^{r+1} i) \quad \forall i, j = 1, ..., n \quad i \neq j \tag{4.5}$$

where 4.3 models the *at-most* constraint for home games, 4.4 the *at-most* constraint for away games and 4.5 the *no-repeat* constraint.

An example of a distance matrix and a feasible round robin schedule for NL6 can be found in Figure 4.2.

Distance matrix of NL6

$$d = \begin{pmatrix} 0 & 745 & 665 & 929 & 605 & 521 \\ 745 & 0 & 80 & 337 & 1090 & 315 \\ 665 & 80 & 0 & 380 & 1020 & 257 \\ 929 & 337 & 380 & 0 & 1380 & 408 \\ 605 & 1090 & 1020 & 1380 & 0 & 1010 \\ 521 & 315 & 257 & 408 & 1010 & 0 \end{pmatrix}$$

Round robin schedule

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

Figure 4.2: Left: The NL6 problem instance from [ENT01] shown as a distance matrix. Right: A feasible double round robin tournament schedule represented by a $(2n-2) \times n$ matrix. For every row r, the team t plays against the entry j at (r,t). If j is negative, the game will be played at $|j|$'s venue, if it is positive at i's.

4.1.2 State Space Formulation

For this section we will follow the formulation presented in [FNR20]. In order to represent the solution space of a TTP instance (V,d), we use a state graph. A state graph is a rooted directed acyclic graph. We will, from now, refer to the nodes of this graph as states. It has a source node that we will call root state and a sink node that we will call terminal state. Each path from the root to the terminal represents a feasible schedule for the given instance. The states are organized in $l = 1, ..., n^2 - n$ regular layers that contain the states representing the configurations after the l-th game in addition to layer 0 that contains the root state s_r and layer $n^2 - n + 1$ that contains the terminal state s_t.

Each state will contain the information about the set P_i^s of games, for each team i, that can be played in accordance to the constraints and is represented as a tuple $(\mathbf{M}^s, \mathbf{y}^s, \mathbf{r}^s, \mathbf{p}^s, \mathbf{h}^s, \mathbf{o}^s)$. $\mathbf{M}^s = (M_{i,j}^s)_{i,j \in V} \in \{0,1\}^{n \times n}$ is an incidence matrix that contains the list of games that are still left to schedule. $\mathbf{y}^s = (y_i^s)_{i \in V}$ is a vector that represents the opponents currently forbidden for team i. The forbidden opponents are important to implement the *no-repeat* constraint and to make sure that each team is playing an opponent only twice. $\mathbf{r}^s = (r_i^s)_{i \in V}$ and $\mathbf{p}^s = (p_i^s)_{i \in V}$ are the current round r_i^s and the location p_i^s for team i. Finally, $\mathbf{h}^s = (h_i^s)_{i \in V}$ and $\mathbf{o}^s = (o_i^s)_{i \in V}$ represent the number of home games (h_i^s) and away games (o_i^s) in a row left to play to respect the *at-most* constraint.

For every state s, we define the transition to the state s' as the game $i \to^r j$, i.e. the game played in round r by i at j's venue. There will be $n^2 - n + 1$ transitions, one for each regular layer plus one for the root state. We assign to each state transition a weight $\Delta z(s, s')$, that we define as the sum of the distance that i and j have to travel in order to reach the venue of the game played between state s and s' from the venue where they

played in their last match:

$$\Delta z(s, s') = d(p_i^s, p_i^{s'}) + d(p_j^s, p_j^{s'}). \tag{4.6}$$

A very important aspect of this representation is that we build the solution round by round, meaning that we cannot start a new round before we decide all the games for the current one. The partial schedule will start from the root state and grow in ascending order until it either ends on the terminal state, meaning that the solution we built is feasible, or it reaches a state with no further transitions, representing an partial schedule that cannot be continued and therefore is infeasible. Root and terminal states will have special rounds that are reserved to them, namely $r = 0$ and $r = 2n - 1$. In these rounds every team will be at their home location. The root state will be the following:

$$s_r = \left(M^{s_r}, y^{s_r} = \begin{pmatrix} -1 \\ \dots \\ -1 \end{pmatrix}, r^{s_r} = \begin{pmatrix} 0 \\ \dots \\ 0 \end{pmatrix}, p^{s_r} = \begin{pmatrix} 1 \\ \dots \\ n \end{pmatrix}, h^{s_r} = \begin{pmatrix} U \\ \dots \\ U \end{pmatrix}, o^{s_r} = \begin{pmatrix} U \\ \dots \\ U \end{pmatrix} \right) \tag{4.7}$$

with M^{s_r} being a matrix of ones with diagonal zeros, allowing each team to play against everyone else. y_i^s is -1 when there is no forbidden team for i in state s. The assignment of p^{s_r} and r^{s_r} make sure that each team starts at their own home at round 0. The terminal state will have all the teams at their own home venue as well. To achieve this, transitions to the terminal state are special because they do not correspond to played games but just send every team home. M^{s_t} will be a matrix of zeros because in a successful schedule everyone played all the available games:

$$s_t = \left(M^{s_t}, y^{s_t} = \begin{pmatrix} -1 \\ \dots \\ -1 \end{pmatrix}, r^{s_t} = \begin{pmatrix} 2n-1 \\ \dots \\ 2n-1 \end{pmatrix}, p^{s_t} = \begin{pmatrix} 1 \\ \dots \\ n \end{pmatrix}, h^{s_t} = \begin{pmatrix} 0 \\ \dots \\ 0 \end{pmatrix}, o^{s_t} = \begin{pmatrix} 0 \\ \dots \\ 0 \end{pmatrix} \right). \tag{4.8}$$

Regular transitions from a state s at layer l to a state s' at layer $l + 1$ will be performed by selecting a game between two teams that are left to play in the current round. This means that $r_i = r_j = \min_{i \in V} r_i$. After selecting team i from the list of available teams, we will select a game against a team j from the list of allowed games P_i^s. The game will either be (i, j) or (j, i) depending on the chosen location. After choosing the next game, if there exists a *dead* team, i.e. a team i with $P_i^s = \emptyset$, the solution we are building is not feasible. The order in which we select teams in a specific round is irrelevant as for the model, but it becomes relevant in practice because we do not enumerate the whole search space. For this reason we introduce an ordering for the teams. An ordering is a permutation $\pi \colon V \to V$ for which, when we play a game from $P_{\pi_i}^s$, i and r_{π_i} must be minimal in every state of layer l.

After a transition (i, j) is selected for state s, the new state s' will be very similar to s, but with some differences. In Figure 4.3 we can see an example of how the state s' gets built when playing a game. As we can see, $\mathbf{M}^{s'}$ will be a copy of $\mathbf{M}^s$ except for $M_{i,j}^{s'} = 0$.

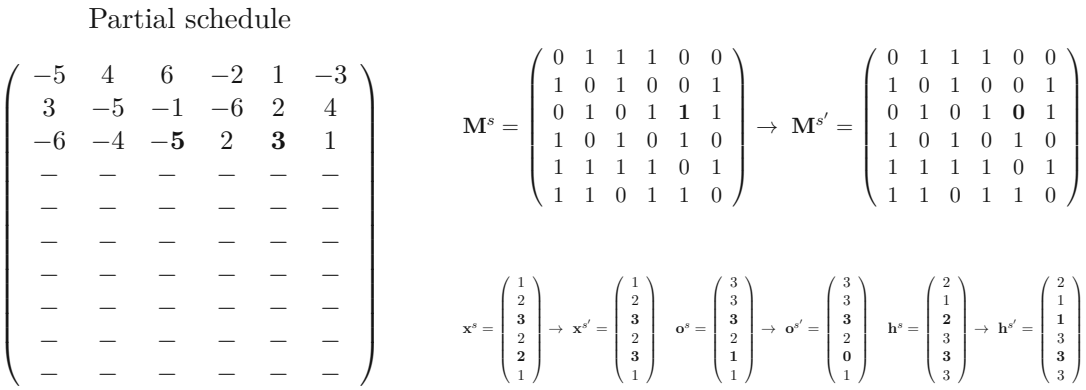

Figure 4.3: Left: Partial schedule of NL6 instance. The next game we select to play is between team three and five at three's venue. The teams appear in bold in the partial schedule. Top right: we update the M matrix, so that the game that we just played is not available anymore. Bottom right: updates of $\mathbf{x}^s$, $\mathbf{o}^s$, $\mathbf{h}^s$. We omitted $\mathbf{r}^s$ and $\mathbf{y}^s$ for space reasons.

This removes i and j from the list of teams available for a game in the current round as well. $\mathbf{y}^{s'}$ will be a copy of $\mathbf{y}^s$ with the following difference: if $M^s_{i,j} = 1$, $y^{s'}_i = j$ and $y^{s'}_j = i$; if $M^s_{i,j} = 0$, $y^{s'}_i = y^{s'}_j = -1$. Furthermore, we set every $y^{s'}_k$ that is equal to either i or j to -1. This way we are making sure of not violating the *no-repeat* constraint. $\mathbf{r}^{s'}$ will be different from $\mathbf{r}^s$ only in the fact that $r^{s'}_i$ and $r^{s'}_j$ will be increased by one. In the same fashion, $\mathbf{p}^{s'}$, $\mathbf{h}^{s'}$ and $\mathbf{o}^{s'}$ will update respectively the current position and the number of home and away games left for team i and team j. These changes make sure to respect the *at-most* constraint: if $h^{s'}_j = 0$, j will be forced to play away from their home venue; similarly, $o^{s'}_i = 0$ will prevent i from playing away again.

4.2 Capacitated vehicle routing problem

We use the Capacitated Vehicle Routing Problem (CVRP) to derive lower bounds for the TTP. In [URM07], Urrutia et al. present a integer programming formulation with for the involved CVRP that we adapt to allow taking an arbitrary state corresponding to a partial schedule as an input to derive a lower bound for the optimal feasible completion of the schedule. In this section we will present said formulation.

4.2.1 Mathematical Model

We now define our vehicle problem $\text{CVRP}_t[s]$ for a given TTP state s and a single team t. For each team t, we are given the set $O_t \subset 2^{\{1,\dots,t-1,t+1,\dots,n\}}$ of the venues still to visit, its position $p_t \in \{1,\dots,n\}$, and its away streak $a_t \in \{0,\dots,U-1\}$. This information can be derived from the t-th row of the incidence matrix $\mathbf{M}^s$ of the games left, the t-th entry of the position vector $\mathbf{p}^s$ and the vector of the remaining games allowed $\mathbf{o}^s$. Furthermore,

p_t will have distance 0 from the depot and no arcs coming from the other teams and demand equal to the streak length a_t. Each team t has to start its schedule from the depot.

The goal is to find the tours of minimum costs so that the team visits all the remaining opponents exactly once with every tour visiting at most U opponents. We define $W_t = O_t \cup \{t, p_t\}$ as the vertices of the graph on which we solve the problem, where the weights of the arcs A_t correspond to the distances between the teams and t corresponds to the depot. Each customer (modeling a remaining venue) has demand equal to one and each vehicle has capacity of U, ensuring that all the remaining venues are visited and that the team does not play more than U away games in a row.

The problem is formulated with the following integer programming model, $\forall t \in \{1, \ldots, n\}$:

$$b_t^{\text{CVRP}}(s) = \min \sum_{i \in W_t} \sum_{j \in W_t} d_{ij} x_{ij} \tag{4.9}$$

subject to:

$$\sum_{j \in O_t} (x_{ij} + x_{ji}) = 2 \quad \forall i \in O_t \tag{4.10}$$

$$\sum_{i \in S} \sum_{j \notin S} (x_{ij} + x_{ji}) \geq 2\lceil |S|/U \rceil \quad \forall S \subset O_t, \tag{4.11}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in A_t \tag{4.12}$$

Furthermore, if $p_t \neq t$, i.e. when team t is currently away, the following constraints must be added:

$$x_{tp_t} = 1 \tag{4.13}$$

In this model, we consider binary decision variables x_{ij}, where $x_{ij} = 1$ if team t travels from the venue of i to the venue of j, otherwise $x_{ij} = 0$. In equation 4.9 we show the objective function for the problem, that returns the minimum distance $dist(t)$ that t has to travel to visit every opponent. Constraint 4.10 ensures that every venue is reached and left only once. Constraint 4.11 makes sure that there is not a tour longer than U venues. Since these are exponentially many constraints, a branch-and-cut solver would add these constraints gradually and only when they are violated. Constraint 4.13 connects the depot to p_t and is only introduced if team t is currently not at its own venue but at p_t.

Finally, the lower bound of the optimal completion of a partial solution in state s is given by the sum of the team bounds of every team:

$$b^{\text{CVRP}}(s) = \sum_{t=1}^{n} b_t^{\text{CVRP}}. \tag{4.14}$$

On Figure 4.4 we show an example of a vehicle routing problem belonging to a single team t to calculate the independent lower bound b^{CVRP} for a given examplary partial solution with corresponding state.

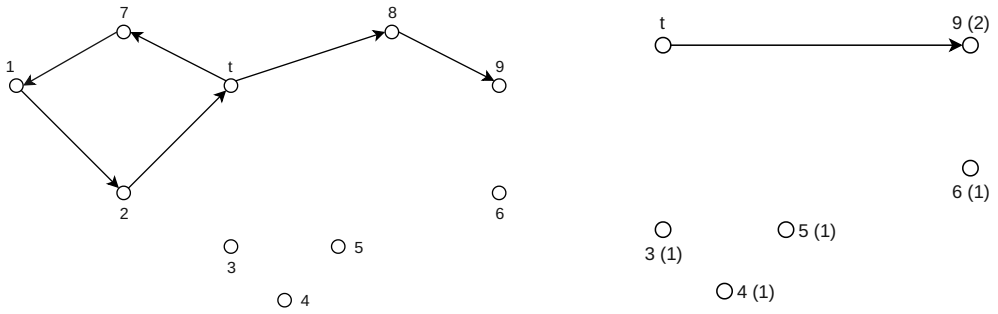

Figure 4.4: Visual representation of our model for CVRP. Left: partial solution where t is in the middle of a tour and is currently at 9's location. Right: we assign $d_{t9} = 0$ and demand of 9 to two.

CHAPTER 5

Solution Approaches

In this chapter, we will present our solution approaches to the TTP and the related CVRP, which we both formalized in Chapter 4. We will start in Section 5.1 by describing our incarnation of an ant colony optimization algorithm for the TTP. It is based on the ACO approach by Uthus et al. [URG09a], with the key difference that no tabu search is applied for each ant but as a novelty a stronger heuristic guidance based on the CVRP lower bound. We solve the latter either by exact precalculation following Uthus et al. [URG09b, URG12] and Frohner et al. [FNR20], or approximately on-the-fly with the help of Google OR-Tools [PF], presented in more detail in Section 5.3. Finally, we describe in Section 5.2 our randomized beam search approach to the TTP, which is based on [FNR20], but uses the approximate CVRP solution as heuristic guidance. This allows to tackle instances with up to 30 teams, where the approach with exactly precalculated bounds is limited to 18 teams. A comparison of our approaches with state of the arts methods from the literature will the be presented in the subsequent Chapter 6, the computational study.

5.1 Ant Colony Optimization Approach

The workhorses of every ACO approach are the virtual ants, which are realized as runs of a randomized construction algorithm. The probabilities of the decisions during the construction change over time as ants communicate the quality of their discovered solutions by means of pheromones. The overall goal is to reinforce good decisions to increase the probability of creating high-quality solutions, while still retaining a certain amount of diversification in the search, to not prematurely converge to a local optimum.

At first, we discuss how ants construct solutions for the TTP by exploring the state space presented in Section 4.1.2. We will then introduce the different heuristic functions used to guide the randomized construction and how the pheromone information is handled and integrated in the algorithm. Finally, we will present further techniques that we used

to improve the performance of the algorithm like backtracking and backjumping to find feasible solutions far more quickly and a restricted candidate list to focus on high-quality extensions during the construction.

In Algorithm 5.1 we list the high-level pseudocode of our ACO algorithm. The procedure receives as first input parameters the number of successful ants N_{ants} and the number of iterations N_{iter}. It then lets the ants construct solutions layer-by-layer, i.e., game by game until N_{ants} feasible solutions have been constructed. We call the group of successful ants a *batch*. After each batch, we select the best solution of the batch as well as the updating ant to update the pheromone. We repeat this process N_{iter} times.

Each ant seeks to create a feasible solution for the TTP. For this, it traverses the state space starting from the initial state s_{r} until it has either found the terminal state s_{t} following the state transition rules as described in Section 4.1.2. It adds games one by one, where each game being played amounts to a state transition. The games are added round per round, meaning that we will not start filling up round r before round $r-1$ is complete. It continues until a feasible solution has been found; in case it hits a dead end, a retraction mechanism has to be employed, where we make use of restarting and backjumping, which will be described in more detail in Section 5.1.3.

An m-partial schedule (or partial solution) is an ordered sequence of m games $(g_1, \ldots, g_m)$, which we generically denote with T. A complete schedule has $n^2 - n$ games. We further keep the list of encountered states during the construction of one solution in a sequence S to allow for backtracking as we will see later. When an ant is at state s, it performs the following actions to select the next game, corresponding to the innermost loop of Algorithm 5.1.

1. It selects a team i for state s by calling the method *next-team(s)*. For each state s there is an ordered list of available teams, i.e., which have not played in current round r (the minimal round over all teams). The ordering is given by a permutation π of the teams fixed in the beginning, either the identity permutation or a random one. *next-team(s)* returns the first element of this list.

2. It calculates the legal games P_i^s for i with respect to the constraints of TTP. For example, if o_i^s is zero, no more away games are allowed. This set is further reduced to P'^s_i by feasibility checks, to detect early that playing a certain game would result in a state without a feasible completion. These checks include

   - whether there are enough home games left for i to accomodate for remaining longest possible away streaks and vice versa away games left for longest possible home stands left, which would lead to a certain *at-most* constraint violation otherwise,
   - whether there are only two games left over all teams concering only two teams, which would lead to a certain *no-repeat* constraint violation,

Algorithm 5.1: $\mathcal{MMAS}$ Ant Colony System for the TTP with Randomized Backjumping and Ant Restarts

Input: number of teams n, distance matrix d, number of ants N_{iter}, number of iterations N_{iter}, pheromone initial value τ_{init}, heuristic function η, heuristic weight α, pheromone weight β, diversification control q_0, RCL fraction α_{rcl}, RCL original weights ω_{rc}, maximum number of backjumps $\zeta_{\max}$, maximum backjump length ζ_{len}, target ants ω_{ants}, pheromone resilience ρ, no improvement limit ψ, pheromone update factor τ_{fac}, probability of applying RCL p_{rcl}, $\mathcal{MMAS}$ p_{best}, pheromone update type τ_{up}

Output: feasible TTP schedule T

$T_{\text{gl}} \leftarrow nil; u_{\text{gl}} \leftarrow \infty$;
$\tau, T_{\text{re}}, u_{\text{re}} \leftarrow$ *restart-ACO*(τ_{init});
for $\iota_{\text{iter}} \leftarrow 1$ **to** N_{iter} **do**
  $\mathcal{B} \leftarrow \emptyset; u_{\text{it}} = \infty$;
  $\tau, T_{\text{re}}, u_{\text{re}} \leftarrow$ *restart-ACO*(τ_{init}) if no impr. of restart-best ant since ψ iterations;
  for $\iota_{\text{ant}} \leftarrow 1$ **to** N_{ants} **do**
    $S, T, s, u \leftarrow$ restart ant;
    while $|T| \leq n^2 - n$ **do**
      $i \leftarrow$ *next-team*(s);
      $P_i^s \leftarrow$ collect permitted games for team i in state s;
      $P'^s_i \leftarrow$ filter P_i^s according to feasibility checks;
      $P''^s_i \leftarrow P'^s_i \setminus \mathcal{B}[s]$;
      if $P''^s_i = \emptyset$ **then**
        blacklist previous game $T.last()$ for state s;
        $S, T, s, u \leftarrow$ restart ant if backjumping limit $\zeta_{\max}$ hit;
        $S, T, s, u \leftarrow$ backjump by random length from $\{1, \ldots, \zeta_{\text{len}}\}$;
      else
        $(j', k') \leftarrow$ *select-game*$(P''^s_i, n, d, s, i, \tau, \eta, \alpha, \beta, \alpha_{\text{rcl}}, q_0, p_{\text{rcl}})$;
        $s', \Delta u \leftarrow$ *play-game*$(s, (j', k'))$;
        $S \leftarrow S \cup s'$; $T \leftarrow T \cup (j', k')$; $u \leftarrow u + \Delta u$;
      end
    end
    make transition to s^{t} by teams which are away go home;
    conditionally update $T_{\text{gl}}, T_{\text{re}}, T_{\text{it}}, u_{\text{gl}}, u_{\text{re}}, u_{\text{it}}$ considering T, u;
  end
  update-pheromones$(\tau, T', \tau_{\text{init}}, u, \rho, \tau_{\max})$;
end
return best found solution T_{gl};

- whether any team would have an empty set of possible games, i.e., whether a dead team would exist.

The first two check are computable in constant time, while the last one is more expensive in $\mathcal{O}(n^2)$ since remaining games for all teams have to be checked.

3. If $P'^s_i = \emptyset$, the ant restarts from the initial state s_r. In Section 5.1.3, a far more efficient method will be presented, namely backjumping, which randomly reverts a number of last games played and "soft" restarts from there to keep a major part of the partial solution.

4. The ant probabilistically selects a game from P'^s_i by performing the *select-game(P'^s_i, n, d, s, i, τ, η, α, β, α_{rcl}, q_0)* procedure listed in Algorithm 5.2. To do so, it uses a mix of heuristic information η^s_{jk} and pheromones τ^s_{jk}, where both indicate how attractive it is to play game (j,k) *while in state* s. Subsections 5.1.1 and 5.1.2 present in detail the implementations of η^s_{ij} and τ^s_{ij}.

   The method *select-game(P'^s_i, n, d, s, i, τ, η, α, β, α_{rcl}, q_0)* receives, amongst others, the parameters α and β which are used to tune the impact of heuristic information and pheromone respectively. A probabilistic weight w_{jk} is calculated for each game $(j,k) \in P'^s_i$, where higher weight indicates a higher likelihood to be selected. Optionally, we restrict the games to the fraction α_{rcl} best (according to w) games, to focus on more promising games. We then randomly select a game (j',k') (either j' or k' is team i) following the ant colony system selection rule with diversification parameter q_0 as presented in more detail in Subsection 5.1.2.

5. Finally, the ant schedules the game between (j',k') by calling the method *play-game(s, (j', k'))* This method performs a state transition to s' as described in Section 4.1.2, copying and updating the state, corresponding auxiliary variables (e.g. the path length to s', $g(s')$), and the partial schedule accordingly with $T \leftarrow T \cup (j',k')$.

When the schedule T is complete, i.e., there are no more games to be played and we are in the last layer of our construction, we send all teams home that played their last game in an away venue, resulting in the terminal state s^t with a solution value of u. In the following sections, we describe in more detail the heuristics to construct the solutions, the pheromone model and update rules, and finally how the randomized construction is boosted.

5.1.1 Heuristic Information

In this section, we will discuss the different functions $\eta^s \colon P'^s \to \mathbb{R}^+_0$ from which we choose one as algorithmic parameter to calculate the probabilistic weights in the *select-game* procedure. Such a function is parameterized by the current state s and maps from the filtered list of games P'^s to a non-negative real value, denoted as η^s_{jk} for a specific game

Algorithm 5.2: *select-game*: Probabilistically selects game for a team

Input: filtered games P'^s_i, number of teams n, distance matrix d, state s, selected team i, pheromone information τ, heuristic function η, heuristic information weight α, pheromone information weight β, restricted candidate list fraction α_{rcl}, diversification control q_0, probability of applying RCL p_{rcl}, weight scaling φ.

Output: next game for i while state s

1 calculate heuristic values η^s_{jk} for each game $(j,k) \in P'^s_i$ using d;
2 calculate probabilistic weights $w_{jk} \propto (\eta^s_{jk})^\alpha \cdot (\tau^s_{jk})^\beta$ for each game $(j,k) \in P'^s_i$;
3 rescale weight $w_{jk} = \varphi(w_{jk})$ or each game $(j,k) \in P'^s_i$;
4 further restrict list of games to fraction of α_{rcl} best games with probability p_{rcl};
5 $(j',k') \leftarrow$ select game using the ant colony system rule with q_0;
6 **return** (k',l');

(j,k). The different function are called *uniform*, Δf, Δg, $\tilde{f}$, and $\tilde{g}$. In the following, we describe those functions by specifying how a single generic η^s_{jk} is calculated, where we denote the resulting state s' (when game (j,k) would be played in state s).

***uniform*:** As the name indicates, each game has the same probability of being selected, i.e., η^s is a constant. If additionally β is set to 0, this is a very simple heuristic method that just selects a random opponent in the list of available ones. This approach acts as baseline for more complex methods, both for running time and quality of results. Furthermore, we use it to test the efficacy of speedups and improvement techniques.

Δf: In Section 3.2 we defined the evaluation of a state s as $f(s) = g(s) + h(s)$, where $g(s)$ is the shortest path to a state s and $h(s)$ is a heuristic estimate for the optimal feasible completion of s. This method calculates the difference between $f(s)$ and $f(s')$, with s being the current state and s' the resulting state after the game in exam is played. Therefore we define Δf as:

$$\Delta f(s,s') = \frac{1}{1 + f(s') - f(s)} = \frac{1}{1 + g(s') - g(s) + h(s') - h(s)}. \tag{5.1}$$

In particular, $g(s') - g(s)$ is the cost for both teams to reach the game location. $h(s')$ and $h(s)$ are heuristic estimate for the optimal feasible completing in the respective states. We derive them from the independent lower bound where for each team a CVRP is solved and summed up. Those bound could either be pre-calculated and looked up in quasi-constant time or a corresponding vehicle routing problem be solved on the fly. We will discuss this lower bound based heuristic in more detail in Section 5.3 in more detail. as a lower bound based heuristic.

We take the reciprocal in order to have a higher value for better results. Since the f-value could be the same for both states (when we make an optimal move), we add 1 to avoid a division by zero.

Δg: The function Δg is defined in a similar fashion as Δf but by assigning the heuristic value $h(s) = 0$ for every state s, i.e., a myopic function. By doing this we greatly decrease the cost of the function at price of precision in the guidance.

$$\Delta g(s, s) = \frac{1}{g(s') - g(s)}. \tag{5.2}$$

$\tilde{f}$: The function $\tilde{f}$ considers only the f-value of the target state $f(s') = g(s') + h(s')$, not the difference as before. Therefore,

$$\tilde{f}(s') = \frac{1}{f(s')}. \tag{5.3}$$

$\tilde{g}$: The function $\tilde{g}$ is the myopic version of f, with $h(s') = 0$. Therefore,

$$\tilde{g}(s') = \frac{1}{g(s')}. \tag{5.4}$$

Note that an actual (expensive) state transition from s to s' does not need to performed but that each of the presented functions allows an incremental evaluation.

5.1.2 Pheromone Model

In this section, we will discuss our pheromone model and the ants' probabilistic selection rule in more detail. We adopt the approach of Uthus et al. [URG09a] and tailor it to our state space model. It combines a $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [SH00] with an ant colony system (ACS) selection rule. We also make use of a three-index pheromone model τ_{jkr}, which represents how attractive it is to play game (j, k) (team j away at k's venue) in round r, where the corresponding pheromone matrix is uniformly initialized to the parameter τ_{init}.

As discussed in Section 5.1, the probabilities of the possible games for team i to play in state s for round r are calculated following the textbook ACO rule as presented in Section 3.3:

$$p_{jk}^{si} = \begin{cases} \frac{(\eta_{jk}^s)^\alpha \cdot (\tau_{jkr})^\beta}{\sum_{(j,k) \in P'^s_i} (\eta_{jk}^s)^\alpha \cdot (\tau_{jkr})^\beta} & \text{if } (j,k) \in P'^s_i \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

This is extended in the ACS [DG97] rule by a parameter $q_0 \in [0, 1]$ which balances between exploitation and diversification. For every choice by the ant, a random number $q \in [0, 1]$ is sampled to randomly switch between a greedy exploitation rule, choosing the game with the highest probabilistic weight, and the aforementioned basic ACO rule to select the next game (j', k'):

$$(j', k') \leftarrow \begin{cases} \text{argmax}_{(j,k) \in P'^s_i} \left[(\eta_{jk}^s)^\alpha \cdot (\tau_{jkr})^\beta \right] & \text{if } q \leq q_0 \\ \text{AS selection rule} & \text{if } q > q_0 \end{cases} \tag{5.6}$$

Following the $\mathcal{MMAS}$, the pheromone values are bound between $\tau_{\min}$ and $\tau_{\max}$ to avoid premature stagnation of the search. We introduced the parameter p_{best}, and $\tau_{\max}$ is set to the asymptotically maximum value of the pheromone, calculated in the following way:

$$\tau_{\max} = \frac{1}{1-\rho}\frac{1}{f(s^{opt})} \tag{5.7}$$

where $f(s^{\text{opt}})$ is the best f-value of the terminal state (i.e., optimum for the instance), for which an estimate needs to be used.

We instead calculate $\tau_{\min}$ in the following way:

$$\tau_{\min} = \frac{\tau_{\max}(1-p_{\text{dec}})}{(\text{avg}-1)p_{\text{dec}}} \tag{5.8}$$

where

$$p_{\text{dec}} = \sqrt[d]{p_{\text{best}}} \tag{5.9}$$

and d is the amount of decisions that an ant has to make, that in our case is the number of games to play $n(n-1)$. The value of avg is $d/2$.

We adopt an elitist approach and make use of restarts (pheromone reinitializiations) as [URG09a]. During the whole run, we keep and update accordingly the solutions and their lengths of the globally best $(T_{\text{gl}}, u_{\text{gl}})$, the best since the last restart $(T_{\text{re}}, u_{\text{re}})$, and the best ant of the current iteration $(T_{\text{it}}, u_{\text{it}})$. To avoid stagnation, we introduce a parameter called ψ that identifies the maximum number of iterations allowed without improvement of the restart-best ant. If this limit is reached, the pheromone matrix is reinitialized to τ_{init}. After a batch of ants is complete in a given iteration, we probabilistically select a solution T' with length u' among the three different ants just described for spreading pheromones according to its solution, a slight modification of the procedure as described by Uthus et al. [URG09a], who perform a deterministic even alternation. The parameter τ_{frac} controls the probability of updating with the iteration-best; if it is not selected, then either the global-best or restart-best ant is selected, depending on another parameter τ_{up}, which can assume the values of "global" or "restart" and is part of the tuning procedure.

To update the pheromones, we call the method *update-pheromone*(τ, T', τ_{init}, u', ρ, $\tau_{\max}$). The pseudocode for it can be found in Algorithm 5.3. The algorithm starts by conditionally updating $\tau_{\max}$ and calculating τ_{min} according to the $\mathcal{MMAS}$ rule, and then setting $\Delta\tau = 1/u$, where u is the shortest path length of the solution used for the update. We then apply the decay according to the pheromone persistence parameter ρ. The auxiliary procedure *games-with-rounds*(T) returns a set of triples, containing all games with the corresponding round. Using it, we finally increase the pheromone value of every played game tied to a round (j, k, r) by $\Delta\tau$, while respecting the bounds $\tau_{\min}$ and $\tau_{\max}$.

Algorithm 5.3: *update-pheromones*: $\mathcal{MMAS}$-updates the pheromone matrix

Input: pheromone information τ, solution T, pheromone initial value τ_{init}, solution length u, pheromone decay rate ρ, maximal pheromone value $\tau_{\max}$

Output: updated pheromone matrix τ

```
calculate τ_min according to MMAS rule;
Δτ ← 1/u ;
τ ← ρ · τ ;
foreach (j, k, r) ∈ games-with-rounds(T) do
    τ_jkr ← τ_jkr + Δτ;
    τ_jkr ← max(τ_jkr, τ_min);
    τ_jkr ← min(τ_jkr, τ_max);
end
```

5.1.3 Randomized Construction Boosting

In our algorithm, we apply some techniques that are aimed at the improvement of either the quality of the solution created or the time taken to produce a feasible solution. Since the TTP is also a difficult constraint satisfaction problem, proper modification to a plain ACO with a purely randomized greedy construction are necessary, since dead ends occur frequently. This is already pointed out by Uthus et al. [URG09a] who use forward checking, conflict-directed backjumping, and ant restart to increase the rate of feasible solution created by an order of magnitude. We also perform forward checking as discussed in Section 5.1 by avoiding games that would leave to team without any permitted games in the given round (empty domain) and a randomized backjumping-based backtracking procedure as discussed in the following.

Backtracking and backjumping: When during our randomized construction a team has no games to play in a round, we reach a dead end. Instead of starting from scratch and lose all of the progress made, we blacklist and revert the last selection made, i.e., we return to the state we were before the last game was played and select either another or backtrack again if there is no other left—this correspond to classical backtracking. The left part of Figure 5.1 shows an illustrative example of a dead end after which we backtrack.

Sometimes backtracking only one step is not enough, because the choice that brought us in a bad part of the search space could have been done several steps before the dead end itself. In such a case, the classical backtracking algorithm would take several steps before realizing it, and would waste a lot of resources exploring a non promising part of the search space. For this reason we use a randomized backjumping prodecure. Backjumping is a well-known technique from constraint programming [Dec90, DF02] and similar to backtracking, but it reverts several steps instead of only one. This allows us to move further from a dead end, to allow to take back a mistake that was made somewhat earlier

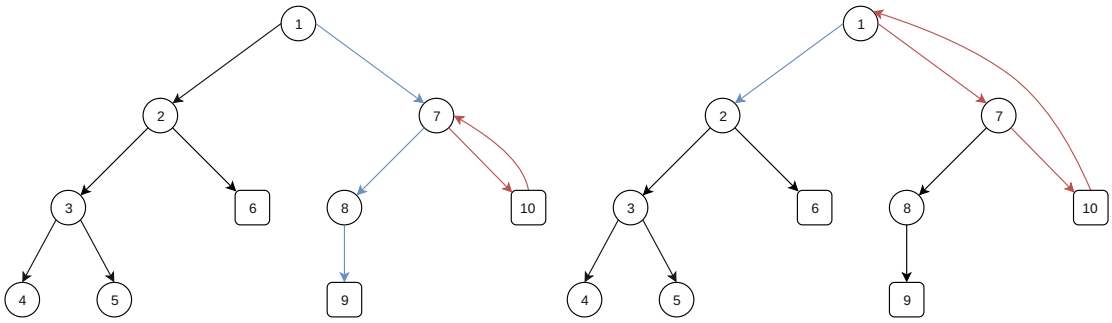

Figure 5.1: Illustrative examples of backtracking and backjumping. Rectangular nodes represent dead ends. Left: After reaching state 10, since it is a dead end we backtrack to 7 and blacklist the action leading to 10. We then choose 8 and proceed from there. Right: After reaching a dead end at 10, we jump back to state 1 and by chance continue with state 2 (again 7 would also have been possible, since the action leading from 1 to 7 is not yet blacklisted), leading us to a better region of the state space.

in the search. An illustrative example of backjumping can be found in the right part of Figure 5.1.

In our implementation we make use of a naive (yet effective as we will see in the computational study) randomized backjumping on our state graph and we have two parameters to tune it, see the corresponding lines 13 to 16 in Algorithm 5.1. The first parameter is $\zeta_{\max}$, and it controls the maximum number of times we retrace our steps before giving up and restart our construction from beginning. Since the search space of TTP is massive, sometimes a partial solution is compromised and it could be better to just abandon it and start a new one: this $\zeta_{\max}$ ensures that we do not spend too many resources on a single solution. The other parameters is ζ_{len} and controls the upper limit of steps we retrace when we reach a dead end. In particular, the former controls the maximum number of steps to revert when we reach a dead end, which we sample uniformly at random from $\{1, \ldots, \zeta_{\text{len}}\}$. Traditional backtracking corresponds to $\zeta_{len} = 1$.

To facilitate backjumping a *blacklist* $\mathcal{B}$ is maintained in form of a hash table with the states of our state graph as keys and as corresponding values the sets of blacklisted games. While future iterations could also benefit from the blacklist, it would grow indefinitely and we reset it after every iteration to keep the memory demand bounded.

Weight scaling: The purpose of weight scaling is to map the probabilistic weights between a maximum and a minimum value in order to limit the capability of strong elements to overwhelm the others, i.e., a case of a degenerate probability distribution. We therefore define a linear transformation $\varphi(w) = \alpha_{\text{s}} \cdot w + \beta_{\text{s}}$ where the minimum weight is mapped to 1 and the maximum weight (corresponding to the most promising game in

a state) is set to a parameter $\varphi_{\max}$. This leads to

$$\alpha_s = \frac{\phi_{\max} - 1}{w_{\max} - w_{\min}} \tag{5.10}$$

and

$$\beta_s = \frac{w_{\max} - \phi_{\max} \cdot w_{\max}}{w_{\max} - w_{\min}}. \tag{5.11}$$

See its use in the procedure *select-game* in Algorithm 5.2, where weight scaling can be disabled by setting φ to the identity function.

Restricted Candidate List: An RCL is a restriction of the set of possible games to a fraction of the most promising ones according to the metric, i.e., only games (j, k) are kept, for which:

$$w_{jk} \geq w_{\max} - \alpha_{rcl} \cdot (w_{\max} - w_{\min}) \tag{5.12}$$

Afterwards, a game is selected randomly from the RCL, where keep the original weights in performing this selection. α_{rcl} is a real value between 0 and 1 controlling the diversification/intensification. When $\alpha_{rcl} = 0$ we select only from the opponent with highest weights (ties are possible), while if $\alpha_{rcl} = 1$ the RCL is the same as the original candidate list. This is well-know technique know from GRASP [RR03]. To make it useful for ACO and keep the whole search space in principle accessible, we introduce another parameter $p_{\text{rcl}} \in [0, 1]$, which represents the probability of applying an RCL in the precedure *select-game*.

5.2 Fast Memory-Limited Randomized Beam Search

The randomized construction described until now builds one solution at the time corresponding to a sequence of games to be played. The rough idea of beam search is to perform a truncated breadth-first-search on the state graph. Instead of storing only one state at a given layer, we store many promising ones and return in the end the shortest path from the root state to the terminal state, i.e., the best found solution. Constructing a solution via beam search would obviously be computationally much more intense than doing so with an ant, but it helps to avoid situations in which we play a game that looks very promising but eventually yields a not so good result. Furthermore, we can tolerate to run sometimes into dead ends without performing backtracking since we keep multiple options to continue to the next layer. In this section we present our extended version [FNPR21] of the beam search approach to the TTP from [FNR20], with the focus on the *approximate* CVRP based lower bound guidance using Google OR-Tools, which will be discussed in the subsequent section.

Beam search is, in essence, a layer-by-layer breadth-first-search traversal of the state graph. In the TTP, for each layer we select a team i and perform the transition from

every existing state to every successor state that is allowed by the list of i's available games, i.e., we play for each state all of i's permitted games leading to *alive* states, employing the same feasibility check to detect dead teams as described for the ACO in Section 5.1. The currently shortest path length and sequence of games that led to a given state are cached for each. Given the complexity of the problem and the size of the search space, the number of states that we keep in consideration by layer is limited by the *beam width* β. Hence, the number of states considered is polynomially bounded by $\mathcal{O}(n^2\beta)$. This alone does not guarantee polynomial asymptotic runtime in n since the expansion and evaluation of states could be $\mathcal{NP}$-hard.

To decide which state is worth keeping, we rank them according to a f-value akin to the one generally used in A* search and that we presented in Chapter 3, where a heuristic estimate to complete the current solution $h(s)$ is added to the length $g(s)$ of the shortest known path to reach the current state s:

$$f(s) = g(s) + h(s) \tag{5.13}$$

We consider a fast memory-limited beam search variant in Algorithm 5.4. The current layer is stored in a queue, and the successive layer in a priority queue of size at most β sorted by the f-value. It is implemented as a binary max-heap combined *without* a hash map that keeps the current position of a state to allow for faster perculation operations. To still enable detection of duplicate state, we keep for each layer a set of seen states when retrieving them from the queue. Since the queue is sorted by the f-value, we can safely discard already seen states (to avoid traversal of isomorphic subgraphs of the state graph), which must have an f-value at least as good. Keeping only the current and last layer gives rise to the adjective *memory-limited*.

Said priority queue H contains the β best successors of all the states in the current layer. If the beam is full, to prevent the eventuality of creating solutions that would be worse than any other solution in the heap, we check by incremental evaluation if the f-value of the last element of the beam is better of the f-value of the state we are about to create. If so, we will not consider the resulting state further, reducing computational effort, otherwise we perform a state transition to s' and incorporate it into the beam.

Calculating the heuristic estimate $h(s)$ of a state may be expensive. Since we base our heuristic guidance on the independent lower bound, where ($\mathcal{NP}$-hard) CVRPs are solved for each team separately in a given state, we maintain a cache of such solved problems. In practice, we observe a very high cache efficiency, since teams are often in the same state over different TTP states of a layer and we can reuse the cached solutions frequently. Another option is to precalculate all possible heuristic estimate for each team, as done in [URG09b, URG12, FNR20]. An approximate and an exact method for solving said CVRPs will be discussed in Section 5.3.

The aforementioned optimality and feasibility checks when in state s and considering game (j, k) are performed by the procedure *feasibility-and-optimality-check*$(H, \beta, s, b, \varepsilon, (j, k))$ in an incremental fashion, avoiding unnecessary costly state transitions. To this end, we

Algorithm 5.4: Fast Randomized Beam Search for the TTP. Adapted from [FNR20], p. 8 and [FNPR21]

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

queue $Q \leftarrow \{s^{\mathrm{r}}\}$;
for $l \leftarrow 1$ **to** $n^2 - n$ **do**
  $\mathcal{S} \leftarrow$ empty set for seen states;
  $H \leftarrow$ empty maximum heap;
  while $Q \neq \emptyset$ **do**
    $s \leftarrow Q.pop()$;
    discard s if $s \in s \in \mathcal{S}$ and continue or $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$;
    $i \leftarrow$ next-team(l, s);
    foreach $(j', k') \in \{(j, k) \in P'^{s}_{i}$ **do**
      $\varepsilon \leftarrow \mathcal{N}(0, \sigma_{\mathrm{rel}}(l) \cdot h(s^{\mathrm{r}}))$;
      if *feasibility-and-optimality-check*$(H, \beta, s, h, \varepsilon, (i, j))$ **then**
        $s' \leftarrow$ copy s and make transition by playing (j', k') and updating state along with cached data accordingly;
        $s'.T \leftarrow s'.T \cup (j, k)$;
        $f(s') \leftarrow g(s') + h(s') + \varepsilon$;
        include s' into H respecting $f(s')$;
        if $H.size > \beta$ **then**
          remove worst element of H;
        end
      end
    end
  end
  $Q \leftarrow$*sorted-by-f-value*(H);
end
if $Q \neq \emptyset$ **then**
  create going home transitions for all states in Q to s^{t};
  return $s^{\mathrm{t}}.T$;
else
  return *no feasible schedule found*;

also cache the number of home and away games left, and the number of teams that have just hit their streak limit. This enables quicker checks regarding the *at-most* constraint and whether there is a team without any permitted games, when we would play game (j, k) in s, i.e., the dead team check.

To enable some degree of diversification in different runs, we introduce a noise variable

and consider a static random team ordering. This variable is a normally distributed random variable with standard deviation σ that we add to the original f-value of each state:

$$\tilde{f}(s) = f(s) + \mathcal{N}(0, \sigma). \tag{5.14}$$

It allows states that would otherwise get pruned to have a chance to survive and vice versa. Since multiple success states may also have the same f-value, it also serves as a random tie breaker. It is important to choose a reasonable value for σ in order to prevent much worse solutions to survive. We use a fraction σ_{rel} of the heuristic estimate of the root state, to ensure that we can pinpoint the order of magnitude of σ.

$$\sigma = \sigma_{rel} \cdot h(s^r) \tag{5.15}$$

The algorithm terminates at the last layer where there are no more games to be played in each state; we then create the transitions to the terminal state, by sending every team that played their last game in an away venue back home and keep and return the resulting best found schedule tied to the terminal state $s^{\mathrm{t}}.T$. If we did not reach the terminal state, we did not find a feasible solution and return this information.

5.3 Capacitated Vehicle Routing Problem

Generally speaking, to obtain a lower bound for a problem it is common to consider a relaxed version of it, for example ignoring a constraint. A classical example is to relax the integer constraints in integer linear programming to obtain a faster solvable linear programming relaxation.This concept was applied to TTP by Easton et al. in [ENT01], where they suggested the independent lower bound (ILB). ILB considers schedules for all teams independently and therefore implicitly relaxes both the *no-repeat* and the at-home part of the *at-most* constraint. In other words it only considers the away streaks. Therefore ILB is, in essence, a series of capacitated vehicle routing problems for every $i \in V$ where the vehicles have capacity U, each customer has demand of 1 and the depot is at i's home venue. As we discussed in Section 2.2.1, CVRP is $\mathcal{NP}$-hard but tractable in practice when the number of customers is sufficiently low. Specifically, we use this lower bound as the basis for our heuristic estimate of a state $h(s)$ for both ACO and beam search. The CVRP based lower bound $b^{CVRP}(s)$ is defined as:

$$b^{CVRP}(s) = \sum_{i=1}^{n} b_i^{CVRP}(s) \tag{5.16}$$

where, for a state s and a team t, we calculate $b_t^{CVRP}(s)$ as the CVRP for the remaining away teams $\mathcal{M}_t^s$, the position p_t^s and the remaining away streak o_t^s. If t is at an away location and the away streak is equal to zero, we add the distance between p_t^s and t to the bound, and then we set $p_t^s = t$ and $o_t^s = \min(|\mathcal{M}_t^s|, U)$.

Given the significant cost of calculating these bounds, following Uthus et al. [URG12], we precalculate the bounds of the possible states to speed up our algorithm. We apply this technique to beam search as Frohner et al. did in [FNR20]. To calculate the bounds we recursively enumerate the states of the whole solution space of each CVRP. For each state we then calculate the shortest path to the terminal state in a backward sweep. Since the solution space is a directed acyclic graph and we store the calculated results in a lookup table accessible in constant time, this operation is linear in the number of vertices and arcs and feasible for instances up to 20 teams.

For instances with more than 20 teams, this precalculation becomes too long to perform, since the number of bounds is in $\mathcal{O}(n^3 2^n)$ [FNR20]. It is important to notice that since the number of state expansions in the beam search is polynomially bounded, it is possible to adopt an on-the-fly approach, calculating the bound approximately only when needed using Google OR-Tools and then storing them independently for each team in a global cache. Moreover, during a transition we only need to calculate the CVRP for two teams, while the others can be retrieved from the cache, and also the states of either or both of these two teams might already been encountered in another TTP state before.

Google OR-Tools [PF] is a software suite tuned for tackling some of the most common optimization problems such as vehicle routing, scheduling and bin packing. The solver allows different methods to find the initial solution for a given CVRP. Here we briefly present the ones that we tested:

- Path cheapest arc: Iteratively, starting from an opponent, attempts to continue a streak until the streak limit is hit by selecting the closest opponent to the last added opponent and adding it to the streak;
- Local cheapest insertion: Inserts each opponent at its cheapest position;
- Global cheapest arc: Iteratively connects the two opponents that produce the cheapest arc;
- Local cheapest arc: connects the first opponent with an unbound successor to the opponent that produce the cheapest arc;
- First unbound min value: connects the first opponent with an unbound successor to the first available opponent.

The algorithm allows to limit the number of solutions generated during the search with the parameter *solution-limit*. After selecting an initial solution, a local search is performed using one of the following methods, which we use as a blackbox:

- Guided local search: escapes local minima and plateaus with a penalized cost function;

- Greedy descent: accepts neighboring solutions only if they have a better cost, until a local minimum is reached;
- Simulated annealing: uses an evolving temperature parameter to balance exploration and exploitation in a guided random walk through the search landscape;
- Tabu search: escapes local minima allowing the acceptance of the best solution from the current neighborhood, avoiding cycles and guiding the search using a restriction mechanism.

In Chapter 6 we will present a series of tests that we performed on the different construction and local search methods provided by Google OR-Tools. Moreover, we implemented a faster version of it in Julia as also presented in [FNPR21], of which we will present the results.

CHAPTER 6

Computational Study

Our experiments were conducted on the cluster of the Algorithms & Complexity Group [1] at the TU Wien with Intel Xeon E5-2640 processors with 2.40 GHz in single-threaded mode. When not stated differently, the memory limit was set to 8GB. The algorithms were implemented in Julia 1.6.0 [BEKS17], interfacing Google OR-Tools v7.7.7810 [PF] with PyEval.jl [2]. Our implementation is based and a continuation of the TTP beam search code[3] [FNR20, FNPR21].

We separate this chapter in four sections. In the first we will present the instances that we used to perform our tests. The second investigates on the usage of Google OR-Tools to solve CVRPs and the impact of different parameters on the beam search. In particular we compare different configurations of solution construction and improvement strategies. We then present the results yielded by beam search guided by OR-Tools for galaxy instances of size 4 to 30 with a fixed beam width. We will also compare the solution quality gap with varying beam widths over a number of instances to get an impression of the impact of an increasing beam width. In the third section we perform the tuning of our ACO approach. More specifically, we will study the impact of the RCL and of different heuristic strategies on the randomized greedy construction and find suitable values for backjumping and backtracking to create feasible solution quickly. Furthermore, we will then show how the solution quality improves over time during an ACO run with different settings of the heuristic and pheromone weight parameter α and β. We will finally discuss the results provided by irace [LIDLP$^+$16], a tool for automatic algorithm configuration and compare them with our manually tuned configuration. In the last section we will compare our ACO with the other ant approaches from the literature, and finally compare our best results for beam search and ACO with the best known results on the considered benchmark instances, derived by other state-of-the-art solvers.

[1] https://www.ac.tuwien.ac.at/
[2] https://github.com/JuliaPy/PyCall.jl
[3] https://github.com/nfrohner/ttpbeam

6.1 Instances

There is a list of instances commonly used as benchmark available on Michael Trick's TTP web page [4]. Instances of NL, NFL and CIRC were first introduced by Easton et al. [VHV07] and they model, respectively, schedules for the Major League Baseball, for the National Football League and for matrices with trivial circular solution for TSP. The Super set of instances is a model of the Super 14 Rugby League and was introduced by Uthus et al. [URG09b] in 2009. In 2012 [URG12] they also introduced the galaxy instances, that model the three dimensional distances in light years that separate stars. In our work we will also use two sets of instances $\mathcal{I}_{L^2}^{\{8,10,12\}}$ and $\mathcal{I}_{L^2}^{\{18,20,22\}}$ which have groups of 30 artificially generated instances using Euclidean distances, where teams are placed uniformly randomly on 1000×1000 integer grid and where each group has a different number of teams as indicated by the superscript. These instances are used by Frohner et al. in [FNR20, FNPR21] for parameter tuning and as a separate training, for which we will use them as well. The set of instances that we use are listed in Table 6.1, together with a description of their origin and the minimum and maximum instance size available for it.

Table 6.1: List of considered TTP benchmark instances with corresponding sizes.

Name	Based on	Min	Max
NL	Major League of Baseball	4	16
NFL	National Football League	16	32
galaxy	Distance between stars	4	40
CIRC	Matrices with trivial circular solution for TSP	4	20
Super	Super 14 Rugby League	4	14
$\mathcal{I}_{L^2}^{\{8,10,12\}}$	Euclidean instances small	8	12
$\mathcal{I}_{L^2}^{\{18,20,22\}}$	Euclidean instances large	18	22

When referring to a specific instance, we will call it with the name and the size (e.g., NL14 is an instance of the instance set NL of size 14).

6.2 Lower Bound based Heuristics using Google OR-Tools

As discussed earlier in chapter 5.3, we believe that Google OR-Tools could be used to solve the CVRPs heuristically on-the-fly with appropriate caching for guiding the beam search fast enough and with a reasonable optimality gap. This is of particular interest

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

for larger instances starting from 20 teams where is not possible to precalculate the bounds exactly. In this section we will show the process of tuning that we performed on OR-Tools as well as first beam search integration tests.

6.2.1 OR-Tools Method Tuning

The results analyzed in this section will also be presented in our submitted journal publication [FNPR21]. The experiment was run on 90 $\mathcal{I}_{L^2}$ instances of size 18, 20 and 22 (30 for each size). Using Gurobi 12.8 [GO21] we solve the classical independent lower bound (ILB) (i.e., $h(s^r)$, the heuristic of the root state for each instance) to optimality for each one of these instances. We then compare the relative optimality gaps and the running times of different construction heuristics and search methods of Google OR-ToolsAs construction heuristics we consider *path cheapest arc* (PCA), *local cheapest insertion* (LCI), *global cheapest arc* (GCA), *local cheapest arc* (LCA) and *first unbound min value* (FUMV), while the search methods are *guided local search* (GLS), *greedy descent* (GD) , *simulated annealing* (SA) and *tabu search* (TS), which we discussed briefly in Section 5.3.

To start, we investigate what is the more suitable CVRP construction heuristic for Google OR-Tools. Each method is tested with solution limit of 20, 30, and 100 with the exception of *greedy descent* that just converges to a local minimum and terminates naturally. The solution limit is the number of iterations (steps in the neighborhood) performed by the search. Treating Google OR-Tools mostly as a black box, we leave the other parameters to default because we assume that they would already perform acceptably.

In Figure 6.1 we present the results of our tests as boxplots that compare relative gaps and running times. As expected, increasing the solution limit yields slightly better gaps at the cost of a much higher running time (y is in logarithmic scale). Since we have to solve many CVRPs during a run of the beam search, runtime is a major concern, potentially at the cost of losing some guidance quality. Furthermore, since we consider a randomized beam search variant, this might still be acceptable.

GCA results are good in terms of gaps but it is also the slowest one, which is what we would expect since it iterates over all arcs in each iteration. We observe that LCA dominates every other method except for GCA, and is in runtime-wise only dominated by FUMV, which has a over poor solution quality performance. Therefore LCA is the method that we deem the best in our situation. Regarding the search methods, both *guided local search* and *tabu search* provide good results but their running times are quite high. On the contrary, the running time of *simulated annealing* is much shorter, but at the expense of gap quality. *Greedy descent* has median gaps slightly above GLS and TS, but the running times are very fast, and therefore this is our preferred choice

6.2.2 Beam Search with OR-Tools

After tuning Google OR-Tools, we incorporate it into our beam search as heuristic guidance. In this section we perform three tests: In the first we run beam search guided by the two most promising search methods, *greedy descent* and *tabu search*, to confirm

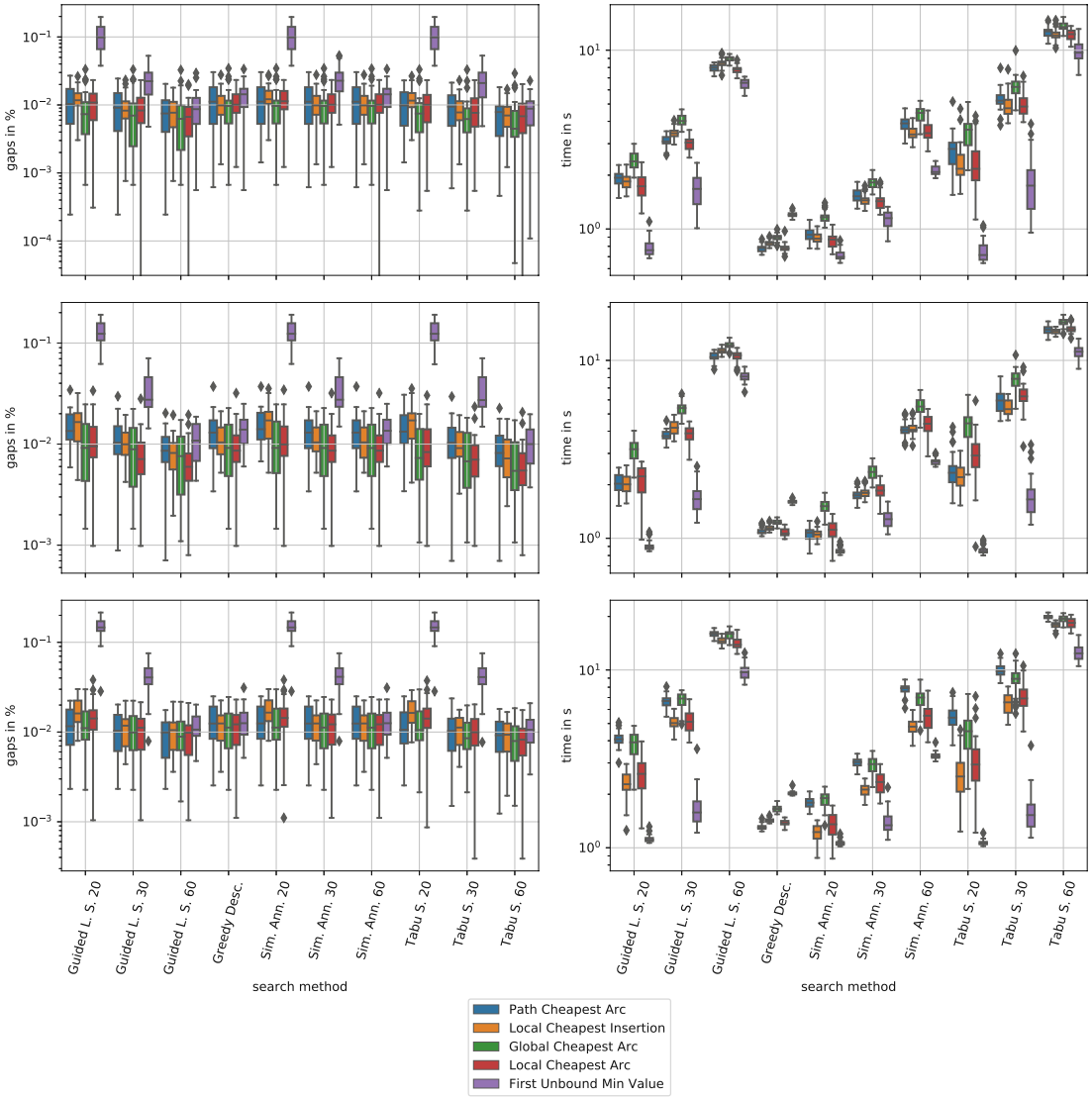

Figure 6.1: Boxplots for relative gap and running times comparison for different construction methods and search method grouped by the random Euclidean instances with team sizes 18, 20, and 22, solving the corresponding root ILB. In the left column we compare relative gaps in %, in the right running times in seconds. The two graphs in the first row describe the testing on instances with 18 teams, while in the second row we have the ones with 20 and in the third with 22. The y-axes are in logarithmic scale. For every search strategy there are five first solution strategies identified by different colors. As expected, higher solution limits yield better gaps but make running time substantially longer. For all of the instances, *guided local search* and *tabu search* provide slightly better gaps but have a longer runtime, compared to *simulated annealing* and *greedy descent*.

the considerations maturated from the tuning of OR-Tools. In the second we run beam search guided by OR-Tools with LCA and GD on three instance varying the *beam-width* parameter β, to see how it effects running times and quality of solutions. In the third we are interested in showing how the running time changes over increasing the size of the instance.

First experiment—greedy descent vs tabu search To further investigate the subtle difference between *greedy descent* and *tabu search*, we ran another test on 90 $\mathcal{I}_{L^2}$ instances of size 18, 20 and 22 (30 for each size), arbitrarily setting the solution limit of TB to 45 so that the objective values is in the same ballpark of the ones of GD. As can be seen in Table 6.2, the two search methods perform very similarly in terms of mean, but there is a substantial difference in running time, with *greedy descent* being consistently faster. After performing an unpaired t-test, the two-tailed p-value of the objectives is higher than 0.84 for all three instance sizes, while the one for runtimes never exceeds 0.001, making the runtime difference extremely statistically significant, while the gaps are not and therefore comparable. For this reason we confirm *greedy descent* as our preferred choice.

Table 6.2: Comparison of objective values and runtimes of greedy descent and tabu search on three instances $\mathcal{I}_{L^2}$.

Instance	search method	objective mean ± std	runtime mean ± std
$\mathcal{I}_{L^2}^{18}$	Greedy descent	157426.50 ± 14017.31	14807.22 ± 1762.36
	Tabu search	158091.64 ± 13889.89	16760.78 ± 503.98
$\mathcal{I}_{L^2}^{20}$	Greedy descent	195345.17 ± 13777.47	20328.87 ± 4568.76
	Tabu search	194628.50 ± 14264.99	23939.93 ± 890.27
$\mathcal{I}_{L^2}^{22}$	Greedy descent	238054.80 ± 18001.28	27795.52 ± 6852.14
	Tabu search	238179.07 ± 16826.68	32478.89 ± 2582.16

Second experiment—impact of β: In the first experiment we run beam search with Google OR-Tools on NL14, circ12 and galaxy16. We set no noise and set β to 10^3, $2 \cdot 10^3$, $5 \cdot 10^3$, 10^4, $2 \cdot 10^4$, $5 \cdot 10^4$, 10^5, $2 \cdot 10^5$ and $5 \cdot 10^5$. The available memory for these tests was 32GB. The goal is to get an impression of the impact that the latter parameter has on solution quality and its cost in terms of construction time. The solution quality is measured in the relative gap between the difference between the shortest path of the solution found and the independent lower bound (ILB) of the instance.

As can be seen in Figure 6.2, when β increases the gaps tend to decrease, but in a rather unpredictable way. A theoretical bound for the construction times with constant n is $\mathcal{O}(\beta \log \beta)$ due to the comparison based sorting of the success states we perform—the logarithmic growth of the slope in β can be surmised in the galaxy16 run, the other runs look noisier. It is impossible to pinpoint a value β that is better than the others, as

it is a tradeoff between solution quality and cost (both in terms of time and memory). Therefore the most reasonable strategy is to use the highest width that we can afford.

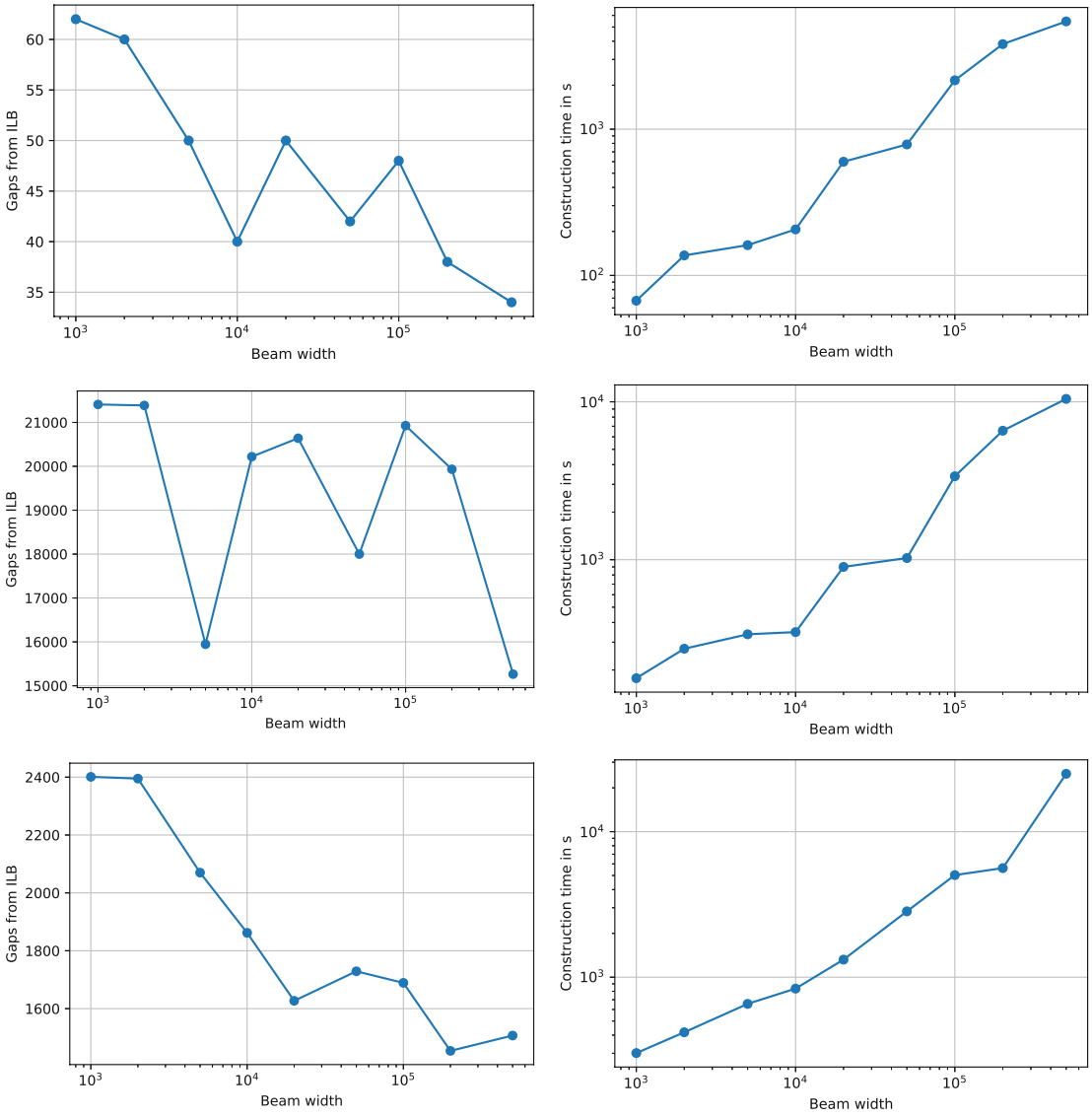

Figure 6.2: Line plots that show the impact of β on beam search with OR-Tools on circ12 (first row), NL14 (second row) and galaxy16(third row). Left: comparison of gaps from the independent lower bound. The x-axis is in logarithmic scale. Higher beam is not necessarily better, but there is a trend towards higher beam widths, in NL14 there is more noise than for the other instances. Right: comparison of construction times in seconds. The x-axis and the y-axis are in logarithmic scale. The theoretical growth depending on β with constant n is bound by $\mathcal{O}(\beta \log \beta)$ when using comparison-based sorting.

Third experiment—running time of different sized instances: We perform this test on a wide range of instance sizes from the set, the smaller being galaxy4 the biggest galaxy32. The beam width β is set to $5 \cdot 10^5$, the largest one from the first experiment. Due to the huge search space of the problem, large beam widths are desirable. Instances of size 26 or larger were run with 32GB of memory instead of 8GB. As can be seen from Figure 6.3, the construction time seems to grow sub-exponentially in the size of n: as expected the construction time grows when the instance size is bigger, where a theoretical bound *without* considering Google OR-Tools is $\mathcal{O}(n^2(n^3\beta + n\beta\log(n\beta)))$ ($\mathcal{O}(n^2)$ layers where in each layer $\mathcal{O}(\beta n)$ successor states have to be considered within $\mathcal{O}(n^2)$ plus comparison-based sorting of the successor states). In reality, we have effects of dead ends reducing the number of expanded nodes, fast optimality and feasibility cuts on the successor states by incremental evaluation, Google OR-Tools heuristic solving for CVRPs depending on n, caching of these heuristics with high cache efficiency, etc. which make a theoretical runtime analysis much more difficult. An exceptionally lucky run happened on galaxy32 which is faster than the run on galaxy28. For example, an instance of size 20 took approximately 72 minutes, while one of size 24 took 165 minutes.

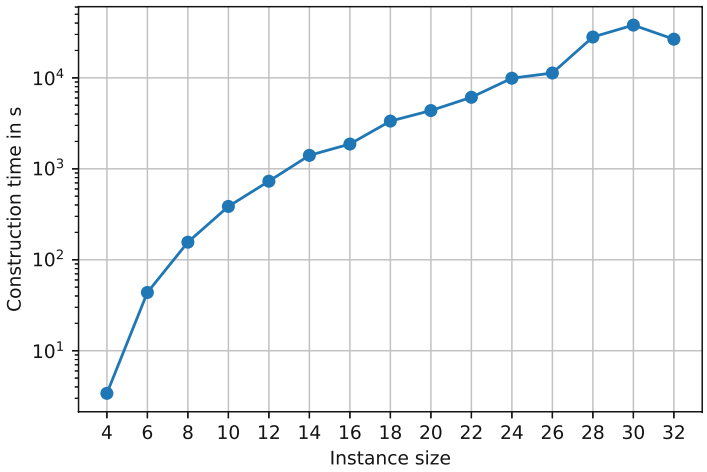

Figure 6.3: Line plot that presents the construction times in seconds of beam search guided by the CVRP bound solved with OR-Tools over instances of different sizes. The x-axis shows the size of the instance, while the y-axis (in logarithmic scale) the construction time in seconds. The construction time spikes up with instances of size greater than 26, with 32 being smaller than 30. The runtime seems sub-exponential but is hard to analyze theoretically due to many competing effects (caching, incremental evaluation, Google OR-tools runtimes, etc.)

6.3 Ant Colony Optimization

In this section we discuss our implementation of the Ant Colony Optimization (as described in Section 5.1) and the experiments that we performed. We will start with the

tuning of the parameters of RCL, backtracking and backjumping parameters, to follow with an analysis on the construction time of solutions on different instance sizes. We perform a manual tuning by making sensible parameter choices and study the impact of different heuristic and pheromone weight combinations on solution quality over time for examplary ACO runs, to observe the learning. We will finally use irace [LIDLP$^+$16], an automated parameter tuning tool, to select a good configuration of parameters for our algorithm.

It is important to note that the ACO relies on precalculated CVRP heuristics since due to its randomized greedy construction and more diverse search through the state space the caching efficiency is too low for the Google OR-Tools to be efficient.

6.3.1 RCL

Firstly, we perform preliminary checks on the impact of using an RCL and the different η ("local heuristics" in an ACO) methods on our algorithm before adding the pheromone information τ. We run the algorithm to create 2000 feasible solutions of NL14 with $\alpha = 1$and β=0, therefore excluding the impact of the pheromones. α_{rcl}, the percentage of solutions retained in the RCL, was tested for the values of 0.10, 0.25, 0.50, 0.75, and 1.00. We keep the weights after restriction and do not select uniformly at random as often done in GRASP. We also tested every heuristic function η, i.e.,Δf, Δg, $\tilde{f}$, *uniform* and $\tilde{g}$. The probability of using RCL p_{rcl} was set to 1. We cross-tested all the combinations of the aforementioned parameters to investigate which effect different RCL-heuristic function combinations have on the solution construction time and quality.As can be seen from Table 6.3, Δg is the method that yields the better results overall, which we also observed for other instances, specifically in conjunction with $\alpha_{rcl} = 0.25$. As we will see later, this does not hold when combining with pheromone information. It also seems to impact the runtime, since we observe that it becomes more difficult to find feasible solutions, which is also true when using other methods than Δf (and not yet fully understood effect)—we therefore abandon it. Still, in a pure construction randomized greedy construction with potentially local search afterwards (i.e., GRASP).

6.3.2 Backtracking and Backjumping

Secondly, we examined the effect of *backtracking* and *backjumping*, as well as to find a good set of parameters for them, to have a high rate of feasible solution construction.

We run our tests on five NL instances from size 6 to 14. We used the *uniform* random method with $\alpha = 1$ and $\beta = 0$, and we do not make use of weight scaling or RCL. The maximum number of backtrackings allowed $\zeta_{\max}$ is a value from 1 to 1000 divisible by 10 and the size of the backjump ζ_{len} goes from 1 to 27. The experiment consists in constructing 1000 ants with each combination of $\zeta_{\max}$ and ζ_{len} as we are interested in comparing the running times in order to find the combination that is able to construct solutions faster.

Table 6.3: Results of experiments on RCL and heuristic methods to create 2000 feasible solutions of NL14. $\min u$ is the best solution, while $\bar{u}$ is the average value. t is the running time relative to the fastest setup. Smaller value of α_{rcl} produce better results at the cost of higher running times. Δg is the heuristic method that yields the better solutions, followed by Δf. *uniform* performs substantially worse than the other methods, which is an unguided random search. Δf has the second best running time.

η	α_{rcl}	$\min u$	$\bar{u}$	t_{rel}
Δf	0.25	244628	269744.63	2.67
	0.50	244356	269929.74	3.11
	0.75	247130	271287.00	2.14
	1.00	254600	275995.00	2.04
Δg	0.25	236509	263200.28	19.62
	0.50	237581	262894.38	9.12
	0.75	241575	266633.09	9.01
	1.00	253898	281692.86	4.01
$\tilde{f}$	0.25	252094	286581.30	32.92
	0.50	260295	290933.36	44.32
	0.75	266915	295152.80	47.52
	1.00	267231	304078.51	1.28
uniform	0.25	291738	314979.46	1.00
	0.50	290452	314912.09	1.69
	0.75	289160	315021.46	1.17
	1.00	287426	315077.56	1.35
$\tilde{g}$	0.25	243117	279800.09	73.50
	0.50	255376	284901.52	110.64
	0.75	266114	290283.81	112.89
	1.00	261963	299637.86	2.89

In Figure 6.4 we present five heatmaps, one for each instance, that show the runtimes of the various combination of parameters. At first glance it is pretty clear that the instance size does not play a substantial role: in fact, in every heatmap there is a slower (bright) region with very high $\zeta_{\max}$ and low ζ_{len}, which broaden slightly with increasing n. As can be seen in more detail in the bottom part of Figure 6.4, the area that presents the highest density of dark dots is the one with $\zeta_{\text{len}} > 15$ and $\zeta_{\max} > 760$, but there is not a particularly strong advantage in selecting one specific setup. We therefore deem appropriate any value of $\zeta_{\max}$ around 1000, while ζ_{len} should be in the range between 20 and 30.

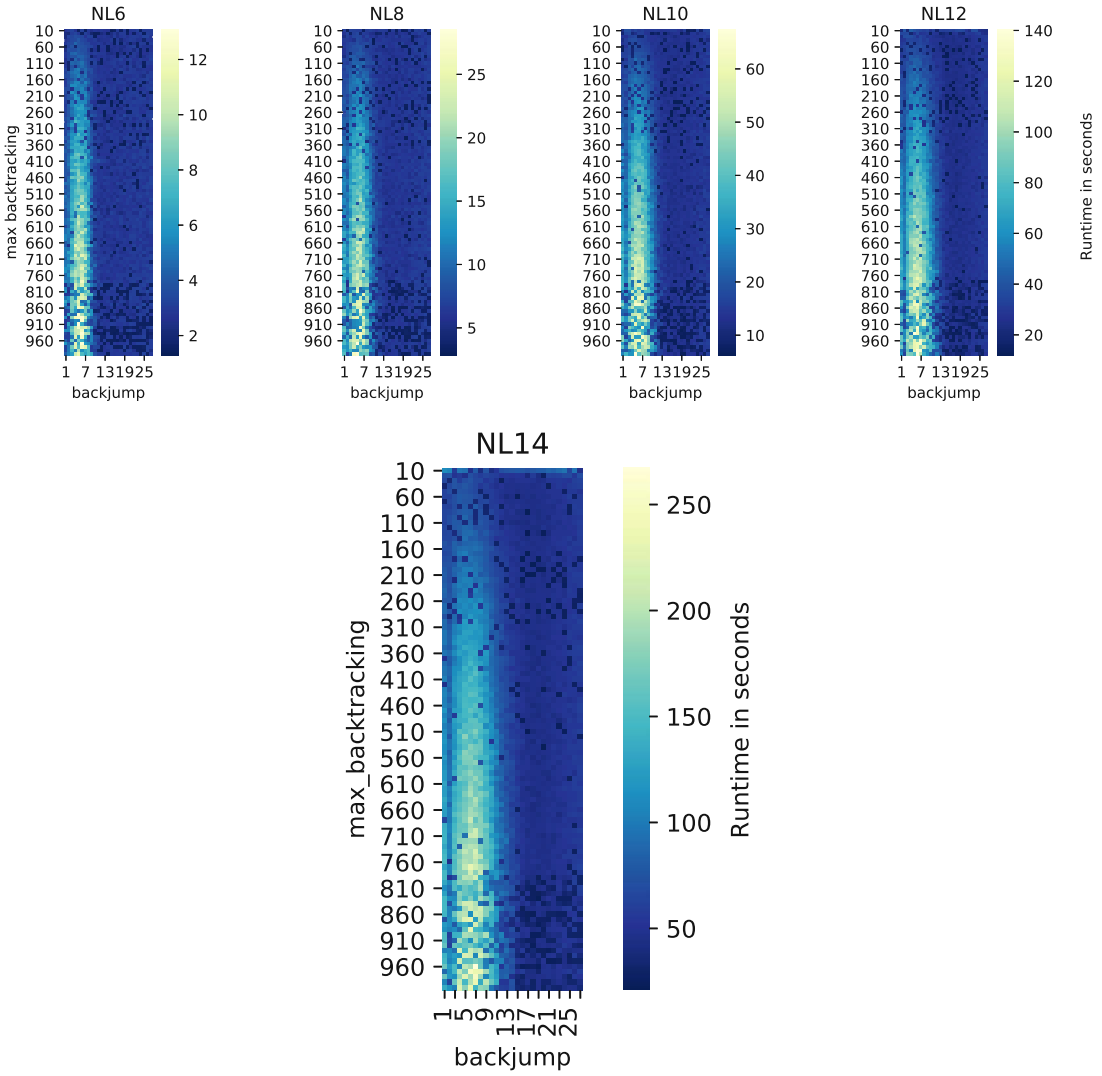

Figure 6.4: Heatmaps for backjump parameter tuning on NL instances of sizes from 6 to 14. The x-axis represents the maximum length of the backjump ζ_{len} with is randomly sampled and the y-axis the maximum number of backtrackings allowed $\zeta_{\max}$. The color represents the time needed to create 1000 feasible solutions. Lighter colored areas have higher running times, while darker dots identify faster setups. Top: runtimes in seconds for instances from 6 to 14. The results look quite similar for all of the instances. Bottom: runtime in seconds for instance of size 14. When $\zeta_{\text{len}} < 15$ the running time spikes up significantly. The higher concentration of black dots can be found with $\zeta_{\text{len}} > 15$ and $\zeta_{\max} > 760$. There are good setups even with $\zeta_{\max} < 300$, but the density is much lower.

6.3.3 Ant Construction Time

In this experiment we analyze the construction time of solutions in instances of different size. We run ACO over NL instances of size 6 to 16, using handpicked parameters selected by preliminary tests. The goal of the experiment is to run 1000 iterations of ACO and measure the average construction time of a feasible solution. We run batches of 30 ants, meaning that we will construct a total of 30000 solutions. For this test, we set the pheromone resilience rate to ρ to 0.9, diversification contl q_0 of 0.9 and update the pheromones with the best ant of the iteration. In Figure 6.5 we can see that the construction time exhibits exponential growth with the size of the instance n with a factor of $0.55n$; for example, a solution for NL8 is constructed in 0.002 seconds on average, while one for NL16 takes 0.14 seconds.

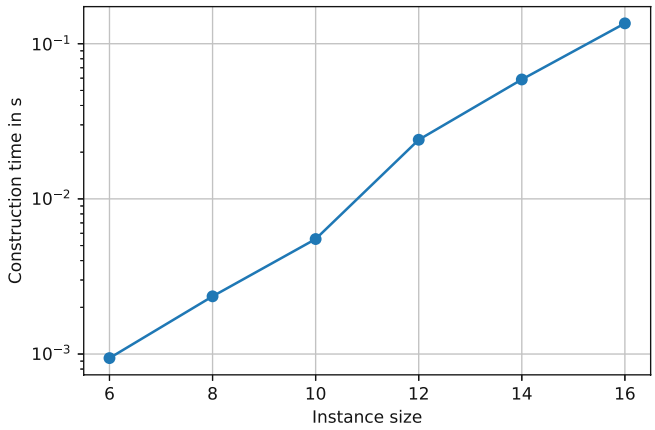

Figure 6.5: Line plot of construction time of feasible solutions over different instance sizes using backtracking with backjump and ant restarts The tested instances were taken from the NL set and they had from 6 to 16 teams. The construction time exhibits exponential growth an estimated exponent of $0.55n$.

6.3.4 Impact of Heuristic and Pheromone Information

Since our algorithm takes into accounts two main values, namely the heuristic information η and the pheromone information τ, it is important to make sure that each piece of information is playing a role and does not get overshadowed by the other. In this experiment we want to analyze the impact of η and τ, as well as how the solutions found by our algorithm improve along the run. A minor goal for the experiment is to compare the effect of using the ACS rule with finite diversification parameter q_0 vs without, i.e. $q_0 = 0$, which we also call AS rule (ant system rule) here. We execute ACO with a set of handpicked parameters on a single instance of NL10.

We use $\eta = \Delta f$ without RCL and fitness scaling, $\zeta_{\max} = 1000$ and $\zeta_{\text{len}} = 30$. The ant rule is ACS with the exception of the third experiment, with $q_0 = 0.8$ and $\rho = 0.8$. We have a

non-improvement iteration limit ψ of 200. We vary the values of α and β to check the interaction between the two parameters. We first negate the effect of the pheromone by setting $\alpha = 1$ and $\beta = 0$. Next, we attempt a balanced setup in which both parameters are set to 1 using ACS and one using AS. The last experiment is run with $\alpha = 0$ and $\beta = 1$, to isolate the pheromone information.

In Figure 6.6 we can see how the heuristic information allows us to start from a good baseline and the pheromone is responsible for the improvement. When we perform our selection with only one of the two methods, the solution quality is clearly inferior. Among the parameter setups that we tested, the better and more consistent results were given by $\alpha = 1$ and $\beta = 1$ with ACS.

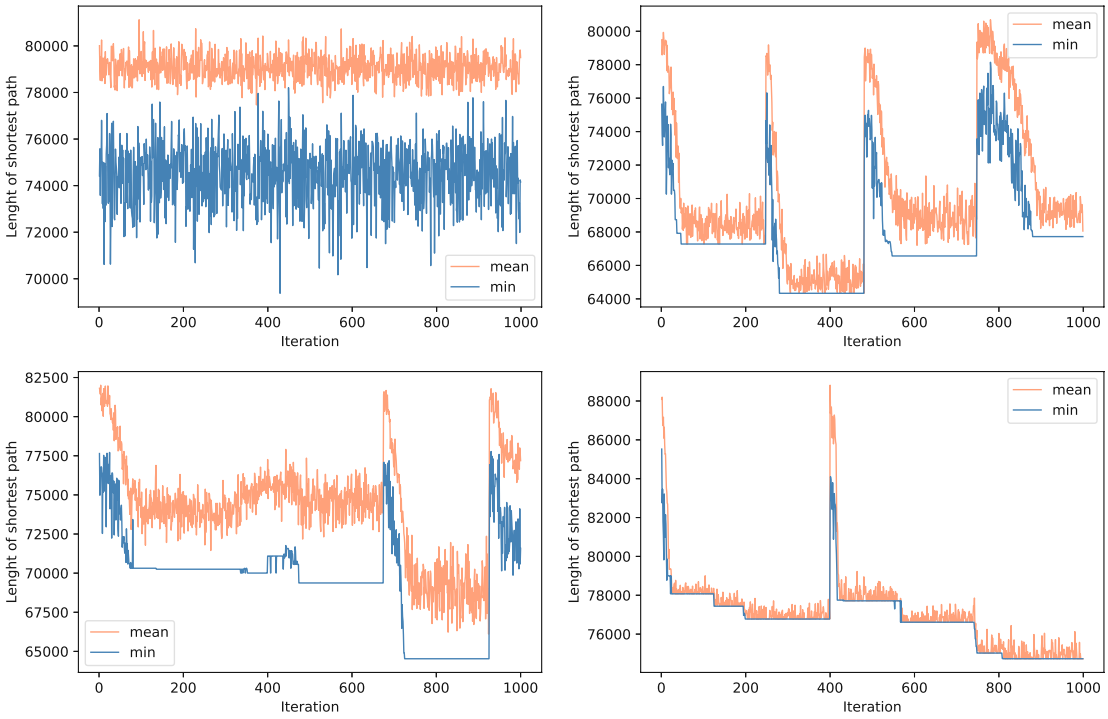

Figure 6.6: Line graphs that show the mean value (orange line, mean solution values over a batch of ants) and the minimal value (blue line, iteration best ant solution value) for every iteration on a NL10 instance. We run ACO for 1000 iteration with 20 ants per batch. Top left: $\alpha = 1$, $\beta = 0$; without any pheromone information, both the mean and the minimal values are stable across the iterations. Top right: $\alpha = 1$, $\beta = 1$, ACS; when η and τ are more balanced, the solution quality is better and more stable. The algorithm reaches a local minimum faster, restarting four times. Bottom left: $\alpha = 0$, $\beta = 1$, AS; when using AS the solution quality improves over time but not as much as ACS, and tends to get stuck more easily in local minima. Bottom right: $\alpha = 0$, $\beta = 1$, ACS; when the heuristic information is absent, the solution quality is very low (the worst of any other test).

Table 6.4: Configuration space for irace and raced elite configurations with a budget of 3000 and 5 irace on the best found solutions for the $\mathcal{I}_{L^2}^{\{8,10,12\}}$ instances over 1000 ACO iterations. For the description of the parameter see Algorithm 5.1.

	N_{ant}	η	α	β	q_0	ρ	ψ	p_{best}	τ_{fac}	τ_{up}
$\mathcal{C}^{\mathcal{I}}$	$\{5,\dots,100\}$	$\{\Delta f, \Delta g, \text{uniform}, \tilde{f}, \tilde{g}\}$	$[0.0, 3.0]$	$[0.0, 3.0]$	$0 \cup [0.1, 0.9]$	$[0.5, 1.0]$	$\{50,\dots,250\}$	$[0.0005, 0.5]$	$\{0.0, 0.25, \dots, 1.0\}$	$\{\text{re}, \text{gl}\}$
C_1	82	Δf	2.316	1.307	0	0.909	226	0.274	1.0	-
C_2	75	Δf	1.440	0.912	0	0.816	200	0.360	1.0	-
C_3	93	Δf	2.153	1.588	0	0.940	217	0.391	1.0	-
C_4	68	Δf	0.937	0.683	0.482	0.795	69	0.295	1.0	-
C_5	90	Δf	2.198	1.505	0	0.952	227	0.297	1.0	-

6.3.5 Automated Parameter Tuning

Due to complexity of the configuration space of the ACO, we also perform an automated tuning of the parameters with irace [LIDLP+16] as cross check for our manual tuning. We train with a budget of at most 3000 runs and 5 iterations with the best found solution over 1000 ACO iterations as metric. As first training instances we use the random Euclidean instances $\mathcal{I}_{L^2}^{\{8,10,12\}}$, being separate instances from the benchmark instances from literature. As an exception, we further train specifically on the circular benchmark set due to their special structure where teams are placed on a circle and are unit distance away from each other, where is also include weight scaling into the configuration space.

We reduce the configuration beforehand based on our manual tuning, to bias irace towards what we believe to be more promising regions of its search space. For instance, the restricted candidate list did not work well together in preliminary tests with ACO, which we therefore disabled. We set the backjumping length to $\zeta_{\text{len}} = 30$ and the backtracking limit to $\zeta_{\text{max}} = 1000$ as tuned before.

The configuration subspaces along with the five elite configurations $C_1, C_2, C_3, C_4, C_5 \in \mathcal{C}^{\mathcal{I}}$ and $C_6, C_7, C_8 \in \mathcal{C}^{\text{CIRC}}$ are shown in Table 6.4 and Table 6.5 respectively. We observe that in the given setting it is beneficial to use Δf along with a stronger heuristic weight α then pheromone weight β, which is consistent over all elite configurations. In $\mathcal{C}^{\mathcal{I}}$, a larger number of ants and the classic selection rule with intensification is preferred ($q_0 = 0$), except once, whereas in $\mathcal{C}^{\text{CIRCI}}$ a higher intensification with the ACS rule and smaller number of ants is selected. Due to the introduced bias, the ACO iteration limit of 1000, and the neglect for runtime in the parameter search, the tuning needs to be taken with a grain of salt. Using the ACS rule (finite q_0) with fewer ants is faster than a configuration where more ants are used with the more diverse AS selection rule ($q_0 = 0$). In the first case, likely due to the intensification and faster convergence, also a smaller no-improvement limit ψ (around 100), where in a latter it is tuned towards ≥ 200. What also meets the eye is that in all but one elite configuration we only use the iteration-best ant for updates, indicating a desire for diversification.

Table 6.5: Configuration space for irace and raced elite configurations with a budget of 1400 and 3 irace on the best found solutions for the CIRC instances over 1000 ACO iterations. For the description of the parameter see Algorithm 5.1

.

	N_{ant}	η	α	β	q_0	ρ	ψ	p_{best}	τ_{fac}	τ_{up}	ϕ_{max}
$\mathcal{C}^{\text{circ}}$	$\{5,\ldots,100\}$	$\{\Delta f, \Delta g, \text{uniform}, \bar{f}, \bar{g}\}$	$[0.0, 3.0]$	$[0.0, 3.0]$	$0 \cup [0.1, 0.9]$	$[0.5, 1.0]$	$\{50,\ldots,250\}$	$[0.0005, 0.5]$	$\{0.0, 0.25, \ldots, 1.0\}$	$\{\text{re}, \text{gl}\}$	$\{0, 10, 20, 50, 100, 1000\}$
C_6	25	Δf	2.384	0.445	0.842	0.77	90	0.181	0.5	re	1000
C_7	39	Δf	1.686	1.151	0.718	0.902	134	0.253	1.0	-	1000
C_8	31	Δf	2.431	0.808	0.701	0.598	87	0.106	1.0	-	0

6.4 Result Comparison

In this section we will compare our ACO approach with other ant approaches state-of-the-art methods on a small set of instances. We then analyze how the results yielded by our beam search with CVRP and ACO approach compare to the absolute best values found in the literature on a big set of popular benchmark instances.

In Table 6.6 we compare our results to the ones obtained by other ACO approaches from the literature. Namely, our first benchmark method will be the one presented by Crauwels and Van Oudheusden in [CVO03], where they apply a basic first version of ACO to TTP with local search and classic backtracking and a two-index pheromone model. We also compare with the one presented in [CKB07] by Chen et al., in which they use ACO as a hyper-heuristic. The last benchmark method that we use is the one presented in [URG09a] by Uthus et al., on which our approach is also partially based. The latter is the most successful ACO algorithm in the literature. In our ACO-CVRP approach, we make use of our manually tuned configuration with a runs of 3000 iterations with 30 ants, with $q_0 = 0.8$, $\rho = 0.8$, $\tau_{fac} = 0.8$, $p_{best} = 0.05$, and τ_{up} set to *restart*. As exception, we use for the CIRC10, CIRC12, and CIRC14 the configuration C_6 (see Table 6.5) which improves the performance on those instances substantially. We use backjumping with $\zeta_{\text{max}} = 1000$ and $\zeta_{\text{len}} = 30$, and do not use RCL and weight scaling, which did not show any beneficial contribution in preliminary tests. The information is balanced with $\alpha = 1$ and $\beta = 1$, using Δf as the heuristic function. We restart after 300 iterations with no improvement.

Crauwels and Chen do not state their averages and how much time their approach had available, so we only report these values for Uthus and our results. We perform our comparison on NL instances starting to size 4 to 16. Our results are substantially better then the first two methods, but fall short in comparison to Uthus's approach, that manages to outperform us with much shorter running times. We followed this last approach quite closely in our work, with some crucial differences. The pivotal innovation of our ACO approach lies in the integration between pheromones and heuristic information: it is possible that the heuristic information, even though it indubitably gives guidance to the search, could eventually prevent the pheromone from exploring certain interesting areas of the search space. Another core difference is Uthus's usage of a local improvement method, in particular tabu search. As previously discussed, our ACO converges quite fast and performs several restarts. A hybridization with a fast local search method would

Table 6.6: Comparison of our ACO approach with results from the literature on NL benchmark instances. Our approach used 3000 iterations with 30 ants per iteration here and a convergence iteration limit of 200. u are the solution lengths, t runtimes in seconds. 20 runs have been performed for AFC-TTP-SHORT and ACO-CVRP.

	Crauwels	Chen			Uthus			ACO-CVRP
	min u	min u	min u	$\bar{u}$	$\tilde{t}$	min u	$\bar{u}$	$\tilde{t}$
instance								
nl4	8276	8276	8276.0	8276.00	10.0	8276.0	8276.00	36.185
nl6	23916	23916	23916.0	23916.00	20.0	23916.0	24042.75	244.760
nl8	40797	40361	39721.0	39785.50	40.0	40739.0	41435.60	594.745
nl10	67871	65168	60399.0	61951.30	80.0	62796.0	64368.25	2053.700
nl12	128909	123752	115871.0	118609.10	160.0	121180.0	122671.00	6685.820
nl14	240445	225169	203205.0	208146.70	320.0	207004.0	214931.30	19589.105
nl16	346530	321037	292214.0	296220.75	640.0	295124.0	304252.20	46512.410

likely be very useful to push through the barrier of a purely constructive ACO, which was out of scope of our work. Furthermore, we observed for the CIRC10, CIRC12, and CIRC14 instances that the difference to the best known optimality gap could be halved by proper parameter meeting, therefore we believe there is still some gain possible.

Finally, in Table 6.7 we show how our approaches perform on each the most popular feasible instances from the literature, as documented on the RobinX project [5].

The values best l and best u represent, respectively, the best lower bound and the best found solution in the literature (i.e., upper bound) for the instance. The value of best u^{rel} is calculated as (best$u$/best l) $-$ 1, meaning that when the lower bound is equal to the upper bound, we have an instance solved to optimality, hence an optimality gap of 0. The results for NL, CIRC and NFL are from Van Hentenryck and Vargados [VHV07], with the exception of circ14, circ16, and circ18 that come from Frohner et al. [FNR20, FNPR21], the beam search approach we extend in this thesis. The one for galaxy14 comes from Langford [Lan10], for galaxy22 they are from Hirano, Abe and Imahori, for galaxy22 from Goerik et al. [GHKW14], and for galaxy instances with sizes, 12, 16, 18, and 20 from the related submitted journal paper [FNPR21]. In this test, we use our two approaches. The first one is beam search with CVRP with beam width of 10^5 and Gaussian noise with relative standard deviation 0.001 added to the guiding f-value. Aside from the noise, further diversification is achieved by randomizing the team ordering in each run. For our ACO, we use the same parameters described above. We executed 20 parallel, separate runs for each instance for each one of the two methods. Some of the instances did not manage to find a feasible solution or incurred into memory problems, namely three NFL22 instances, two NFL26, six galaxy22 and one galaxy26.

[5] https://www.sportscheduling.ugent.be/RobinX/travelRepo.php

We index the results generated by beam search with 1 and the ones generated by ACO with 2. For $i \in \{1, 2\}$, min u_i is the best solution for the run, $\bar{u}_i$ is the average and $\bar{t}_i$ is the average time in hours.

The gaps min u_i^{rel} are calculated as (min u_i/best l) $- 1$ and represent how good of a result that is in comparison with the lower bound. The gaps $\Delta \min u_i^{rel}$ are instead comparisons in respect with the best solution of the literature and it is calculated as the relative gap difference between *best* u^{rel} and min u_i^{rel}.

As anticipated in the previous test, ACO does not perform too well, presenting $\Delta \min u_2^{\text{rel}}$ of more than 10% on almost every instance, where we conjecture that a hybridization with a fast local search method is necessary to be competitive. On the other hand, the performance of beam search with CVRP is comparable to state-of-the-art methods, managing to keep a $\Delta \min u_1^{\text{rel}}$ of less than 6% and in the mean of 2.7% for every single instance.

Table 6.7: Comparison with the best known lower bounds best l and upper bounds best u from the literature over the CIRC, galaxy, NFL, NL, and Super benchmark sets of our randomized beam search (20 runs, beam width $1.0 \cdot 10^5$) with random team ordering and Gaussian noise with relative standard deviation of 0.1% of the root heuristic estimate guided by the exact CVRPH bound (solution lengths u_1) and our best performing ACO configuration (solution lengths u_2).

inst	n	best l	best u	best u^{rel}	min u_1	$\bar{u}_1$	$\bar{t}_1[h]$	min u_1^{rel}[%]	Δ min u_1^{rel}[%]	min u_2	$\bar{u}_2$	$\bar{t}_2[h]$	min u_2^{rel}[%]	Δ min u_2^{rel}[%]
nl10	10	59436	59436	0.0	59788.0	60226.2	0.2	0.6	0.6	62536.0	64624.4	0.3	5.2	5.2
nl12	12	108629	110729	1.9	113606.0	114335.6	0.4	4.6	2.6	120462.0	122472.8	0.9	10.9	9.0
nl14	14	183354	188728	2.9	197668.0	201848.7	0.7	7.8	4.9	209079.0	214039.3	2.8	14.0	11.1
nl16	16	249477	261687	4.9	269414.0	275778.5	1.2	8.0	3.1	293127.0	303019.2	6.8	17.5	12.6
nfl16	16	223800	231483	3.4	238410.0	243726.0	1.1	6.5	3.1	261126.0	268710.7	6.7	16.7	13.2
nfl18	18	272834	282258	3.5	296269.0	303252.0	1.7	8.6	5.1	320097.0	342106.1	14.9	17.3	13.9
nfl20	20	316721	332041	4.8	350773.0	359730.0	2.3	10.8	5.9	-	-	-	-	-
nfl22	22	378813	402534	6.3	416639.0	426411.2	3.4	10.0	3.7	-	-	-	-	-
nfl24	24	431226	463657	7.5	472888.0	487505.3	4.4	9.7	2.1	-	-	-	-	-
nfl26	26	495982	536792	8.2	553580.0	570067.2	5.9	11.6	3.4	-	-	-	-	-
galaxy10	10	4535	4535	0.0	4590.0	4616.4	0.2	1.2	1.2	4807.0	4965.9	0.2	6.0	6.0
galaxy12	12	7034	7180	2.1	7351.0	7447.3	0.4	4.5	2.4	7984.0	8154.6	0.5	13.5	11.4
galaxy14	14	10255	10879	6.1	11036.0	11188.0	0.7	7.6	1.5	12148.0	12603.5	1.3	18.5	12.4
galaxy16	16	13619	14648	7.6	15062.0	15183.4	1.1	10.6	3.0	16901.0	17401.6	3.2	24.1	16.5
galaxy18	18	19050	20489	7.6	20947.0	21249.8	1.6	10.0	2.4	23639.0	23957.9	6.9	24.1	16.5
galaxy20	20	23738	25818	8.8	26016.0	26538.2	2.3	9.6	0.8	-	-	-	-	-
galaxy22	22	31461	33901	7.8	35656.0	35963.9	3.3	13.3	5.6	-	-	-	-	-
galaxy24	24	41287	44526	7.8	46318.0	47015.9	4.3	12.2	4.3	-	-	-	-	-
galaxy26	26	53802	58968	9.6	61484.0	61884.6	5.9	14.3	4.7	-	-	-	-	-
circ10	10	242	242	0.0	244.0	246.6	0.2	0.8	0.8	260.0	266.9	0.3	7.4	7.4
circ12	12	388	404	4.1	416.0	420.4	0.4	7.2	3.1	440.0	451.0	0.9	13.4	9.3
circ14	14	588	628	6.8	638.0	646.1	0.7	8.5	1.7	694.0	738.3	2.2	18.0	11.2
circ16	16	846	910	7.6	924.0	941.5	1.1	9.2	1.7	1160.0	1181.3	1.3	37.1	29.6
circ18	18	1188	1284	8.1	1320.0	1342.4	1.6	11.1	3.0	1644.0	1678.3	2.7	38.4	30.3
circ20	20	1600	1732	8.3	1788.0	1816.3	2.2	11.7	3.5	-	-	-	-	-
super10	10	316329	316329	0.0	317593.0	318498.8	0.2	0.4	0.4	323740.0	333936.0	0.5	2.3	2.3
super12	12	453860	460870	1.5	461799.0	467584.2	0.4	1.7	0.2	479476.0	489334.8	1.4	5.6	4.1
super14	14	557354	571632	2.6	578363.0	585190.4	0.7	3.8	1.2	623021.0	649701.6	3.7	11.8	9.2

CHAPTER 7

Conclusions and Future Work

In this work we investigated on the usage of the capacitated vehicle routing problems (CVRPs) as a lower bound based heuristic guidance for solving the traveling tournament problem with constructive approaches. We started by modeling this independent lower bound for a given TTP state and team as CVRP to be solved either exactly by precalculation or heuristically on-the-fly with Google OR-Tools [PF]. Based on the state space formulation as introduced in [FNR20], we proposed and implemented a $\mathcal{MAX} - \mathcal{MIN}$ ACO approach to tackle TTP using the independent lower bound as local heuristic for the ants. We then extended the implementation of a randomized beam search variant to the TTP, so that it can be used with Google OR-Tools to tackle instances with up to 26 teams. We compare these approaches and the effects of different algorithmic parameters on artificial instances and on well-known benchmark instances from the literature.

The results for our ant based method were better than some other older or simpler ant approaches, but still worse than the current best method presented by Uthus et al. [URG09a]. As discussed in Section 6.4 we followed Uthus's approach quite closely with the addition of the heuristic information. The idea behind it is that the heuristic information should serve as guidance to the ants and direct them towards better areas of the search space faster, but in practice the results are not as good as expected. A possible explanation is that the heuristic information is too predominant and after an initial help prevents the pheromone to explore the search space. Another crucial difference with Uthus is our lack of local search: some sort of daemon action may be crucial to overcome the barrier on which ACO stops. Furthermore, our implementation has over twenty five tunable parameters. This makes the algorithm quite challenging to optimize due to the sheer amount of possible combinations; specific techniques (e.g., RCL) could give good results on their own but then not work very well when combined with another technique. Parameter tuning is a possible area of improvement and could be a focus in future works. Another natural improvement for our ACO is to hybridize it with a fast local search

method, or combine it with beam search, where ants create only the first few rounds of a solution to be continued by beam search with reasonable beam width.

In contrast to the moderate results of ACO, beam search guided by CVRP based bounds using Google OR-tools performs comparably to other state-of-the-art methods, reporting relative gap differences to the best known gaps smaller than 6% and in the mean of 2.7% on a set of 28 difficult benchmark instances from the literature.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AFGG11] Claudia Archetti, Dominique Feillet, Michel Gendreau, and M. Grazia Speranza. Complexity of the vrp and sdvrp. *Transportation Research Part C: Emerging Technologies*, 19(5):741–750, 2011. Freight Transportation and Logistics (selected papers from ODYSSEUS 2009 - the 4th International Workshop on Freight Transportation and Logistics).

[AMS05] Claudia Archetti, Renata Mansini, and Maria Grazia Speranza. Complexity and reducibility of the skip delivery problem. *Transportation Science*, 39(2):182–187, 2005.

[AMVHV06] 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.

[ANB+95] Philippe Augerat, D Naddef, JM Belenguer, E Benavent, A Corberan, and Giovanni Rinaldi. Computational results with a branch and cut code for the capacitated vehicle routing problem. 1995.

[BBS+09] Prasanna Balaprakash, Mauro Birattari, Thomas Stützle, Zhi Yuan, and Marco Dorigo. Estimation-based ant colony optimization and local search for the probabilistic traveling salesman problem. *Swarm Intelligence*, 3(3):223–242, 2009.

[BEKS17] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[BH83] John A Barnes and Frank Harary. Graph theory in network analysis. *Social networks*, 5(2):235–244, 1983.

[Bha09] Rishiraj Bhattacharyya. A note on complexity of traveling tournament problem. *Optimization Online*, 2480, 2009.

[BHS99] Bernd Bullnheimer, Richard F Hartl, and Christine Strauss. An improved ant system algorithm for thevehicle routing problem. *Annals of operations research*, 89:319–328, 1999.

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

[Blu05] Christian Blum. Beam-aco—hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591, 2005.

[BVB08] Christian Blum, Mateu Yábar Vallès, and Maria J Blesa. An ant colony optimization algorithm for dna sequencing by hybridization. *Computers & Operations Research*, 35(11):3620–3635, 2008.

[CDMT94] Alberto Colorni, Marco Dorigo, Vittorio Maniezzo, and Marco Trubian. Ant system for job-shop scheduling. *JORBEL-Belgian Journal of Operations Research, Statistics, and Computer Science*, 34(1):39–53, 1994.

[CH97] Daniele Costa and Alain Hertz. Ants can colour graphs. *Journal of the operational research society*, 48(3):295–305, 1997.

[Chr79] Nicos Christofides. The vehicle routing problem. *Combinatorial optimization*, 1979.

[CKB07] Pai-Chun Chen, Graham Kendall, and Greet Vanden Berghe. An ant based hyper-heuristic for the travelling tournament problem. In *2007 IEEE Symposium on Computational Intelligence in Scheduling*. IEEE, April 2007.

[CL07] Christian H. Christiansen and Jens Lysgaard. A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research Letters*, 35(6):773–781, 2007.

[CVO03] H. Crauwels and D. Van Oudheusden. Ant colony optimization and local improvement. In *Workshop of Real-Life Applications of Metaheuristics, Antwerp, Belgium*, 2003.

[CW64] Geoff Clarke and John W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[DCD98] Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[DDC99] Marco Dorigo and Gianni Di Caro. Ant colony optimization: a new metaheuristic. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1470–1477. IEEE, 1999.

[Dec90] Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.

[DF02] Rina Dechter and Daniel Frost. Backjump-based backtracking for constraint satisfaction problems. *Artificial Intelligence*, 136(2):147–188, 2002.

[DG97] Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[DGS07] 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.

[DMC96] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[DS10] Marco Dorigo and Thomas Stützle. Ant Colony Optimization: Overview and Recent Advances. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, pages 227–263. Springer US, Boston, MA, 2010.

[DW80] Dominique De Werra. Geography, games and graphs. *Discrete Applied Mathematics*, 2(4):327–337, 1980.

[ENT01] Kelly Easton, George Nemhauser, and Michael Trick. The traveling tournament problem description and benchmarks. In Toby Walsh, editor, *Principles and Practice of Constraint Programming — CP 2001*, pages 580–584, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[ENT02] 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*, pages 100–109. Springer, 2002.

[FNPR21] Nikolaus Frohner, Bernhard Neumann, Giulio Pace, and Günther R. Raidl. Approaching the traveling tournament problem with randomized beam search. *Evolutionary Computation*, 2021. submitted.

[FNR20] Nikolaus Frohner, Bernhard Neumann, and Günther R. Raidl. A beam search approach to the traveling tournament problem. In Luís Paquete and Christine Zarges, editors, *Evolutionary Computation in Combinatorial Optimization*, pages 67–82, Cham, 2020. Springer International Publishing.

[GD00] Luca Maria Gambardella and Marco Dorigo. An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3):237–255, 2000.

[GHKW14] Marc Goerigk, Richard Hoshino, Ken-ichi Kawarabayashi, and Stephan Westphal. Solving the traveling tournament problem by packing three-vertex paths. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[Glo86] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[GO21] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021.

[GTA99] Luca Maria Gambardella, Éric Taillard, and Giovanni Agazzi. Macs-vrptw: A multiple colony system for vehicle routing problems with time windows. In *New ideas in optimization*. Citeseer, 1999.

[GTD97] Luca Maria Gambardella, Éric Taillard, and Marco Dorigo. Ant colonies for the qap. Technical report, Citeseer, 1997.

[GW16] 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.

[HNR68] 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.

[HR05] Refael Hassin and Shlomi Rubinstein. On the complexity of the k-customer vehicle routing problem. *Operations Research Letters*, 33(1):71–76, 2005.

[HT19] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. *arXiv preprint arXiv:1911.09539*, 2019.

[Irn10] Stefan Irnich. A new branch-and-price algorithm for the traveling tournament problem. *European Journal of Operational Research*, 204(2):218–228, July 2010.

[KGV83] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[Lan10] Glenn Langford. An improved neighbourhood for the traveling tournament problem. *arXiv preprint arXiv:1007.0501*, 2010.

[LD10] Ailsa H. Land and Alison G. Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer, 2010.

[LIDLP+16] 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.

[LLLY10] Chou-Yuan Lee, Zne-Jung Lee, Shih-Wei Lin, and Kuo-Ching Ying. An enhanced ant colony optimization (eaco) applied to capacitated vehicle routing problem. *Applied Intelligence*, 32(1):88–95, 2010.

[Low76] Bruce T. Lowerre. The harpy speech recognition system. *Ph. D. Thesis*, 1976.

[MC99] Vittorio Maniezzo and Alberto Colorni. The ant system applied to the quadratic assignment problem. *IEEE Transactions on knowledge and data engineering*, 11(5):769–778, 1999.

[MJ76] RH. Mole and SR. Jameson. A sequential route-building algorithm employing a generalised savings criterion. *Journal of the Operational Research Society*, 27(2):503–511, 1976.

[MMI12] Ryuhei Miyashiro, Tomomi Matsui, and Shinji Imahori. An approximation algorithm for the traveling tournament problem. *Annals of Operations Research*, 194(1):317–324, 2012.

[NMNP87] H. Ney, D. Mergel, A. Noll, and A. Paeseler. A data-driven organization of the dynamic programming beam search for continuous speech recognition. In *ICASSP'87. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 12, pages 833–836. IEEE, 1987.

[OM88] Peng Si Ow and Thomas E. Morton. Filtered beam search in scheduling. *The International Journal Of Production Research*, 26(1):35–62, 1988.

[ON00] Stefan Ortmanns and Hermann Ney. Look-ahead techniques for fast beam search. *Computer Speech & Language*, 14(1):15–32, 2000.

[Osm93] Ibrahim Hassan Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, 41(4):421–451, 1993.

[PF] Laurent Perron and Vincent Furnon. Or-tools. `https://developers.google.com/optimization/`.

[PR91] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

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

[PS03] Sriram Pemmaraju and Steven Skiena. *Computational discrete mathematics: Combinatorics and graph theory with mathematica®*. Cambridge university press, 2003.

[R+77] D. Raj Reddy et al. Speech understanding systems: A summary of results of the five-year research effort. *Department of Computer Science. Camegie-Mell University, Pittsburgh, PA*, 17:138, 1977.

[RN02] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

[RR03] Mauricio G. C. Resende and Celso C. Ribeiro. Greedy randomized adaptive search procedures. In *Handbook of metaheuristics*, pages 219–249. Springer, 2003.

[RU07] Celso C. Ribeiro and Sebastián Urrutia. Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research*, 179(3):775–787, 2007.

[Sam17] Claude Sammut. *Beam Search*, pages 120–120. Springer US, Boston, MA, 2017.

[SB99] Ihsan Sabuncuoglu and M. Bayiz. Job shop scheduling with beam search. *European Journal of Operational Research*, 118(2):390–412, 1999.

[SH97] Thomas Stützle and Holger H. Hoos. Max-min ant system and local search for the traveling salesman problem. In *Proceedings of 1997 IEEE international conference on evolutionary computation (ICEC'97)*, pages 309–314. IEEE, 1997.

[SH00] Thomas Stützle and Holger H. Hoos. – ant system. *Future Generation Computer Systems*, 16(8):889–914, June 2000.

[SH05] Alena Shmygelska and Holger H. Hoos. An ant colony optimisation algorithm for the 2d and 3d hydrophobic polar protein folding problem. *BMC bioinformatics*, 6(1):1–22, 2005.

[STN94] Volker Steinbiss, Bach-Hiep Tran, and Hermann Ney. Improvements in beam search. In *Third International Conference on Spoken Language Processing*, 1994.

[TV02] Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, January 2002.

[TV03] Paolo Toth and Daniele Vigo. The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on computing*, 15(4):333–346, 2003.

[TW11] Clemens Thielen and Stephan Westphal. Complexity of the traveling tournament problem. *Theoretical Computer Science*, 412(4-5):345–351, 2011.

[URG09a] 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.

[URG09b] David C. Uthus, Patricia J. Riddle, and Hans W. Guesgen. Dfs* and the traveling tournament problem. In Willem-Jan van Hoeve and John N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 279–293, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[URG12] 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, October 2012.

[URM07] S. Urrutia, C. C. Ribeiro, and R. A. Melo. A new lower bound to the traveling tournament problem. In *2007 IEEE Symposium on Computational Intelligence in Scheduling*, pages 15–18, 2007.

[VB96] Alex Van Breedam. *An analysis of the effect of local improvement operators in genetic algorithms and simulated annealing for the vehicle routing problem.* RUCA Belgium, 1996.

[VHV07] Pascal Van Hentenryck and Yannis Vergados. Population-based simulated annealing for traveling tournaments. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, number 1, pages 267–262. MIT Press, 2007.

[ZH05] Rong Zhou and Eric A. Hansen. Beam-stack search: Integrating backtracking with beam search. In *ICAPS*, pages 90–98, 2005.