CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

PHD THESIS

Material handling by means of autonomous mobile robots (AMRs) is a phenomenon that has gained momentum in the last few years, as the perception and decision-making capabilities of the robots increase, and as computers become more powerful and can control larger and larger fleets. In modern industrial applications, fleets of AMRs operate in a heterogeneous environment, shared with humans and other vehicles and obstacles. In this work we model the features of a modern production environment and thus formulate the *Conflict-Free Electric Vehicle Routing Problem* (CF-EVRP). The inputs to the CF-EVRP are:

- information on the fleet (how many AMRs, of what type, and their operating range);
- list of tasks to execute (location in the plant and time windows for execution);
- plant layout (road segments, allowed travelling direction, and depots location).

Solving the CF-EVRP provides a schedule for the fleet of AMRs to execute the tasks within their time windows, and to account for the AMR's limited operating range, so that the charging time at the depots is part of the schedule. Moreover, the CF-EVRP includes capacity constraints on the road segments, limitations on the number of robots that can travel on road segments at the same time.

The overall problem is to find solutions that satisfy all constraints while avoiding travelling unnecessarily long routes, and at the same time meet the stipulated time-windows to deliver material just-in-time. The compositional algorithm (ComSat) presented in this work is based on the idea to break down the overall scheduling problem into sub-problems that are easier to solve, and then to build a schedule based on the solutions of the sub-problems. ComSat is designed to work well for industrial scenarios where there are good reasons to believe that feasible solutions do exist. This is a reasonable assumption as in an industrial setting where a sufficient number of mobile robots can be assumed to be available.

CHALMERS
UNIVERSITY OF TECHNOLOGY

SABINO FRANCESCO ROSELLI • Conflict-Free Routing of Mobile Robots • 2022

Conflict-Free Routing of Mobile Robots

An SMT/MILP-based Compositional Algorithm

SABINO FRANCESCO ROSELLI

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Conflict-Free Routing of Mobile Robots

Sabino Francesco Roselli

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2022

Conflict-Free Routing of Mobile Robots

Sabino Francesco Roselli

Technical Report No. 5174
ISSN 978-91-7905-708-4

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000
www.chalmers.se

Printed by Chalmers Reproservice
Gothenburg, Sweden, September 2022

I don't claim to be a mathematician, but these words inspired my work...
"Mathematics, he said, isn't merely a question of applying rules, any more than science. It doesn't merely make the most combinations possible according to certain fixed laws. The combinations so obtained would be exceedingly numerous, useless and cumbersome. The true work of the inventor consists in choosing among these combinations so as to eliminate the useless ones, or rather, to avoid the trouble of making them, and the rules that must guide the choice are extremely fine and delicate. It's almost impossible to state them precisely; they must be felt rather than formulated."

Robert M. Prising on Henri Poincaré
Zen and The Art of Motorcycle Maintenance
1974

Abstract

The recent advances in perception have enabled the development of more autonomous mobile robots in the sense that they can operate in a more dynamic environment where obstacles surrounding the robot emerge, disappear, and move. The increased perception of Autonomous Mobile Robots (AMRs) allow them to plan detailed on-line trajectories in order to avoid previously unforeseen obstacles, making AMRs useful in dynamic environments where humans, traditional fork-lifts, and also other mobile robots operate. These abilities contributed to increase automation in logistic applications. This thesis discusses how to efficiently operate a fleet of AMRs and make sure that all tasks are successfully completed.

Assigning robots to specific delivery tasks and deciding the routes they have to travel can be modelled as a variant of the classical Vehicle Routing Problem (VRP), the combinatorial optimization problem of designing routes for vehicles. In related research it has been extended to scheduling routes for vehicles to serve customers according to predetermined specifications, such as arrival time at a customer, amount of goods to deliver, etc.

In this thesis we consider to schedule a fleet of robots such that areas avoid being congested, delivery time-windows are met, the need for robots to recharge is considered, while at the same time the robots have freedom to use alternative paths to handle changes in the environment. This particular version of the VRP, called CF-EVRP (Conflict-free Electrical Vehicle Routing Problem) is motivated by an industrial need. In this work we consider using optimizing general purpose solvers, in particular, MILP and SMT solvers are investigated. We run extensive computational analysis over well-known combinatorial optimization problems, such as job shop scheduling and bin-packing problems, to evaluate modeling techniques and the relative performance of state-of-the-art MILP and SMT solvers.

We propose a monolithic model for the CF-EVRP as well as a compositional approach that decomposes the problem into sub-problems and formulate them as either MILP or SMT problems depending on what fits each particular problem best. The performance of the two approaches is evaluated on a set of CF-EVRP benchmark problems, showing the feasibility of using a compositional approach for solving practical fleet scheduling problems.

Keywords: Job Shop, Vehicle Routing, Bin Sorting, SMT, MILP.

List of Publications

This thesis is based on the following publications:

[A] **Sabino Francesco Roselli**, Kristofer Bengtsson, Knut Åkesson, "SMT solvers for job-shop scheduling problems: Models comparison and performance evaluation". Proceedings of 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), Munich, Germany.

[B] **Sabino Francesco Roselli**, Fredrik Hagebring, Sarmad Riazi, Martin Fabian, Knut Åkesson, "On the Use of Equivalence Classes for Optimal and Sub-Optimal Bin Packing and Bin Covering". 2021 IEEE Transactions on Automation Science and Engineering (TASE).

[C] **Sabino Francesco Roselli**, Martin Fabian, Knut Åkesson, "Solving the Conflict-Free Electric Vehicle Routing Problem Using SMT Solvers". Proceedings of 2021 29th Mediterranean Conference on Control and Automation (MED), Bari, Italy.

[D] **Sabino Francesco Roselli**, Per-Lage Götvall, Martin Fabian, Knut Åkesson, "A Compositional Algorithm for the Conflict-Free Electric Vehicle Routing Problem". 2022 IEEE Transactions on Automation Science and Engineering (TASE).

[E] **Sabino Francesco Roselli**, Remco Vader, Martin Fabian, Knut Åkesson, "Leveraging Conflicting Constraints in Solving Vehicle Routing Problems". Proceedings of 16th IFAC Workshop on Discrete Event Systems (WODES) September 7-9, 2022 - Prague, Czechia.

Specification of my contribution to the included publications:

[A] I formulated the models as MILP and SMT and implemented the code, ran the experiments and wrote the manuscript. The co-authors contributed with discussions about the model formulations and the experiments, and with ideas for structuring and formulating the final manuscript.

[B] I, together with my supervisors (and co-authors), formalized the concept of *skinny* and *fit* bins, proved that it can be used to achieve optimal solutions

and developed the heuristic method to achieve sub-optimal solutions. I implemented most of the code, ran the experiments and wrote the manuscript. The co-authors contributed with the script to generate *skinny* and *fit* bins, an additional comparison to a CP solver (not included in the paper), and structuring and formulating the final manuscript.

[C] I defined the CF-EVRP and formulated its mathematical model as SMT; I implemented the code, ran the experiments and wrote the manuscript. The co-authors contributed in the formalization of requirements for the CF-EVRP and with ideas for structuring and formulating the final manuscript.

[D] I developed the main algorithm and formulated the mathematical models for the sub-problems. I implemented the code, ran the experiments and wrote the manuscript. The co-authors contributed with ideas regarding the algorithm and the mathematical models, and helped structuring and formulating the manuscript.

[E] I supervised a master student (second author) on developing the *Unsat Core* guided search, implementing the code, and running the experiments. I wrote the manuscript. The other co-authors contributed with discussions about the *Unsat Core* guided search and helped structuring and formulating the final manuscript.

Other publications by the author, not included in this thesis, are:

[F] **Sabino Francesco Roselli**, Fredrik Hagebring, Sarmad Riazi, Martin Fabian, Knut Åkesson, "On the Use of Equivalence Classes for Optimal and Sub-Optimal Bin Covering". *Proceedings 15th IEEE Conference on Automation Science and Engineering (CASE)* Vancouver, Canada, Aug. 2019.

[G] **Sabino Francesco Roselli**, Kristofer Bengtsson, Knut Åkesson, "SMT Solvers for Flexible Job-Shop Scheduling Problems: A Computational Analysis". *Proceedings 15th IEEE Conference on Automation Science and Engineering (CASE)* Vancouver, Canada, Aug. 2019.

[H] **Sabino Francesco Roselli**, Kristofer Bengtsson, Knut Åkesson, "Compact Representation of Time-Index Job Shop Problems Using a Bit-Vector Formulation". *Proceedings 16th IEEE Conference on Automation Science and Engineering (CASE)* Hong Kong, Aug. 2020.

[I] Elinor Jernheden, Carl Lindström, Rickard Persson, Max Wedenmark, Endre Erős, **Sabino Francesco Roselli**, Knut Åkesson, "Comparison of Exact and Approximate methods for the Vehicle Routing Problem with Time Windows". *Proceedings 16th IEEE Conference on Automation Science and Engineering (CASE)* Hong Kong, Aug. 2020.

[J] **Sabino Francesco Roselli**, Martin Fabian, Knut Åkesson, "An SMT Based Compositional Algorithm to Solve a Conflict-Free Electric Vehicle Routing Problem". *Proceedings 17th IEEE Conference on Automation Science and Engineering (CASE)* Leon, France, Aug. 2021.

[K] Alvin Combrink, Stephie Do, Kristofer Bengtsson, **Sabino Francesco Roselli**, Martin Fabian, "On Generic Constraints for Nurse Rostering: Definition, Modelling and Comparison". *Submitted for Publication to the Journal of Scheduling.* 2022.

Acknowledgments

A wise man (who is also acknowledged here, he knows I am talking about him) once told me that acknowledgements are a crucial part of a thesis, and that everybody who is close enough to the author will look for their names in here, to see if they left a mark in his/her journey to become a doctor.

It is also a way for the author himself to take some time to realize that he was not alone during the trip, but had many people that supported him, in many different ways.

My first thanks go to **Knut Åkesson**, my main supervisor. One of the best memories I have about these five years at Chalmers are the countless hours spent in your office, sketching on the board some strange hieroglyphs, trying to find the right answers, sometimes succeeding, sometimes not. Thank you for everything you taught me, how to be a good researcher, how to write about my findings and let other people know about it, and how to be a good teacher and pass on my knowledge to others.

Martin Fabian my co-supervisor since 2020. Thank you for all the good research we have done together. No mistake or inconsistency will escape your merciless scrutiny, which means much more work for me to fix them, but also much better results. Thank you also for all the fun we had outside Chalmers, at conferences, after-works, or underground bars. I hope I have fulfilled my duties as a *Vice-Dean of Fun*, now it is time for you to select a successor.

Kristofer Bengtsson my co-supervisor and climbing partner. I must have told you this over a beer once or twice, but one of the main reasons I accepted the job at Chalmers, was that you were going to co-supervise me. In the beginning I felt a stranger to this field of robots and computational complexity and I did not think I was up for the task, but you always had the right words to reassure me. Not to mention all the awesome climb we did together ... I take credit for your kids being so good at it now.

The Seniors of the Automation Group, Bengt, Petter, Sahar, and Emmanuel. Your experience and wisdom have been a valuable asset for me to grow as a researcher and guided me when I had to take important steps for my career here at Chalmers.

My colleagues and friends from SysCon and E2. I shared with you most of the hours spent pursuing this PhD. I complained to you about not getting the right results, I celebrated with you my little successes when a paper got accepted or I passed an exam. I took excessively long lunch breaks

with you, especially when the weather was nice and we were sunbathing in the courtyard. You have been more than just colleagues to me, you have been friends! So thank you *Mattias, Constantin, Zahra*, my office mates; thank you *Martin, Fredrik, Sarmad, Ashfaq, Adnan, Nina, and Oscar*, my former colleagues; thank you *Endre, Rickard, Ludvig, Julius, Anton, Johan, Jonas, Yuvaraj, Ze, and Alvin*, the Automationers of E2. Also a big thank to all the researchers from the other groups in SysCon and E2.

My friends in Gothenburg. I spent so much time talking to you about my work, probably boring you with technical details and tricky corner cases, but even so you were always there supporting me and helping me have a good time when I needed to. Thanks ***Rebecca***, *Will, Anna, Kaylie* (probably misspelled), *Bevan, Eden, Linnea, Shelby, Delaney, Francesco, and Fred.*

My Family. Last but not least I want to give the most heartfelt thanks and dedicate this work to my parents *Rosa Anna and Domenico*, my siblings *Raffaele, Lianna and Sandra* and their significant others *Valentina and Giovanni*, and my little nieces *Roberta, Giorgia, Nia, and* (soon) *Flavia.*

This work has been supported by EUREKA ITEA3-projektet ENTOC (Label nr. 15015, Engineering Tool Chain for Efficient and Iterative Development of Smart Factories), Vinnova, Chalmers AI Research Centre (CHAIR), AB Volvo (Project ViMCoR), and ITEA3-projektet AIToC (Artificial Intelligence supported Tool Chain in Manufacturing Engineering). I also thank the support from Per-Lage Götvall at Volvo Group Truck Operation, and the Wallenberg AI, Autonomous Systems and Software program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Acronyms

AMO:	At Most One
AMR:	Automated Moving Robot
BCP:	Bin Covering Problem
BPP:	Bin Packing Problem
BSP:	Bin Sorting Problem
CF-EVRP:	Conflict Free-Electric Vehicle Routing Problem
EN:	Exactly n
EO:	Exactly One
FJSP:	Flexible Job Shop Problem
JSP:	(Standard) Job Shop Problem
MILP:	Mixed Integer Linear Programming
OMT:	Optimization Modulo Theory
SAT:	Boolean Satisfability
SMT:	Satisfiability Modulo Theory
VRP:	Vehicle Routing Problem
VRPTW:	Vehicle Routing Problem with Time Windows

Contents

Part I

Overview

CHAPTER 1

Introduction

The past few decades have witnessed a change of paradigm: the main production resources are no longer human beings, but robots and computers. This is happening not only on the shop floor, but at every echelon of the production. This thesis focuses on solving logistic problems involving a fleet of Autonomous Moving Robots (AMRs). It investigates efficient methods to schedule AMRs with limited operating range, such that they can pick up and deliver goods throughout the plant within given time windows. Moreover, the schedule is such that the AMRs do not incur in *conflicts* with each other, i.e., they do not block each other's way due to geometrical limitations of the paths they are travelling on.

1.1 Automation in Modern Production Facilities

Modern technologies are revolutionizing the industrial world. Machines can execute calculations faster and more reliably than humans. Computers can schedule the production in a matter of minutes, saving hours or even days of pen-and-paper work. Robots are nowadays advanced enough to execute complex manufacturing or assembling tasks [1], thus being able to replace human

workers in an ever growing number of situations. The world is experiencing an unprecedented shift towards automated production.

Another important truth about modern production can be effectively expressed by the following quote:

"Chances are, if something can be expressed as a mathematical equation, then at some point somebody will want to minimize it." [2]

When it comes to modern production systems, different solutions in terms of process design or manufacturing plan can affect some outcome parameters; it could be a direct cost, or the manufacturing time, or it could be environment related, such as overall CO_2 emissions. Therefore, not every solution has the same quality. Companies want to increase their profit and decrease their cost, governments want to reduce pollution. No matter what the parameter to control is, the goal is to find, among all possible solutions, the one that optimizes (minimizes or maximizes) it.

One important task that has drawn the attention of both academia and industry in the last decades is *planning* and *Scheduling* [3], [4]. In a nutshell, planning is the task of deciding what and how much needs to be done, while scheduling is the task of deciding who needs to do it and when [5]. As industrial systems grow larger and more complex, manual planning and scheduling is no longer an option.

However, even for the most advanced computers, computing a plan or a schedule can be challenging. The large size and the complex requirements of modern scenarios have added so many degrees of freedom to the industrial systems, that the number of possible solutions can be in the same order as the number of atoms in the universe, or even larger. Therefore, brute force enumeration is not an option, because it could take up to thousands of years to check all the solutions.

Fortunately, in many cases, brute force enumeration is avoidable. One can reason about the problem and find relations among its elements to develop algorithms that quickly find a solution. For some problems it is actually possible to quickly find the *optimal* solution with respect to a certain parameter by means of these tailor-made algorithms; let us label the problems whose solution can be found quickly as *tractable*.

On the other hand, many problems are not tractable; there exist problems that are proved to be *hard* to solve (further details about this are given in Section 2.1). In general, even for hard problems it is still possible to design

algorithms that are better than brute force enumeration, but such algorithms may still be too slow.

While the definition of *tractable* and *hard* problems will be provided in formal terms in the next chapter, let us now focus on what are the implications when dealing with one class of problems or the other. The key factor in industrial applications is the time available to solve the problem. If a problem is *tractable*, we can assume that no matter what instance of such problem we have to solve, we will have enough time to produce a solution. On the other hand, for *hard* problems, time becomes a constraining factor, and it is imperative to come up with a solution in reasonable time.

When a *hard* problem involves optimization, one option is to trade quality for time [6]. In other words we accept a worse solution, in terms of the parameters to optimize, as long as we can compute it faster. One common strategy when dealing with *hard* optimization problems is *problem decomposition*; relevant decomposition methods are *Lagrangean decomposition* [7], *column generation* [8], and more recently *Benders decomposition* [9]. The original problem (often called *master problem*) is broken down into sub-problems, and so is its complexity. The sub-problems are then iteratively solved and their solutions are used to build a solution to the original problem. This fragmentation of the original problem may cause a loss of information, therefore it is not always possible to guarantee that the solution found will be optimal. However, the sub-problems are easier to solve and even if it takes multiple iterations to achieve a solution to the original problem, the overall computation time can be shortened.

1.2 General Purpose Solvers and Optimization Problems

Solving different problems requires different algorithms. However, there exist more general algorithms that are able to compute solutions for several problems. For instance, Mixed Integer Linear Programming (MILP) [10] can be used to solve problems arising from real-world systems, as long as such systems can be described by linear constraints and objective function. On the other hand, MILP is not the only approach that provides a general framework to deal with linear constraints. Among other approaches, Satisfiability Modulo Theory (SMT) [11] is nowadays a viable alternative and have in quite a few

cases showed interesting performance when dealing with industrial problems, as in [12] and [13].

Over the years, powerful MILP and SMT solvers have been developed [14], [15]. These solvers provide a flexible framework to model many industrial problems. The end user does not need to be an experienced computer scientist to develop an algorithm that will solve the problem. Instead they need to be able to model the problem, if possible, as a set of linear expressions and then feed it to the solver.

Modern solvers can quickly solve large models, counting up to tens of thousands of variables and expressions [16]. Even so, modern industrial problems are sometimes so complicated that these solvers are not able to provide a good enough solution within the available time. For these reasons, a tailor-made algorithm can be designed to solve a specific problem, because by exploiting the problem structure, it can yield a better performance in terms of computation time.

In some cases a hybrid approach can prove very efficient [17]–[20]; an overall algorithm is developed to solve the problem by means of *problem decomposition*, and then the sub-problems resulting from breaking down the original problem are solved by means of general purpose solvers.

1.3 The *Conflict-Free Electric Vehicle Routing Problem (CF-EVRP)*

The main industrial problem tackled in this thesis is called the *Conflict-Free Electric Vehicle Routing Problem* (CF-EVRP). This problem is about computing a schedule for a fleet of AMRs with a limited operating range and the ability to recharge their batteries at the depot. The schedule must be such that the robots are able to deliver goods to customers within given time windows. Also, the schedule must account for geometrical limitation of the road segments the AMRs are travelling on, so that they do not block each other's way.

Material handling by means of AMRs has been gaining momentum for quite some years now [21]. Traditionally, the mobile robots used for logistic operations work in a controlled environment, following predefined lanes and with low to no interaction with other robots or humans. This is so because of the inherent complexity of an ever changing environment such as a manufacturing

or assembly plant, and the limited perception and decision capability of the robots.

This trend is changing [22] as the perception and decision capabilities of the robots increased over the years. The new challenge is to use large fleets of AMRs that can navigate through a maze of corridors, shelves, and workstations, react to changes in the environment, and deliver components in time to ensure the production schedule is not disrupted. As the number of robots increases, and as they are allowed to operate next to humans and other vehicles, so does the possibility of the robots running into static or dynamic obstacles. Therefore the AMRs have to be able to plan their trajectory on-line in order to avoid such obstacles; they also have to be able to avoid conflicts with other AMRs. Beside obstacles avoidance, the AMRs have to manage their battery level in order to be able to execute their task and go back to the charging station; most important of all, they have to meet the time windows for pickup and delivery operations. They cannot be too late in order to avoid disruption in the production schedule, nor can they be too early in order to avoid congestion by the workstations.

The schedule for a system like the one described may need to be continuously updated, because every time a change occurs, the old schedule may become obsolete; if a pallet is suddenly left in the middle of a road, blocking the transit of AMRs in both directions, all the AMRs using that road would have to be re-routed and may then interfere with other AMRs schedules; if an AMR breaks down, its task would have to be assigned to a new AMR.

Every time a change occurs, a new schedule must be computed in order to guarantee continued production. However, computing a schedule can be very challenging; as mentioned before, the number of factors to take into account is high, and they intertwine with one another, creating an even larger number of possible solutions. Moreover, when scheduling the AMRs, minimizing the travelled distance, or the number of vehicles required, or the level of congestion in the plant is usually desired, therefore optimization is required. Finally, the requirement of delivering goods within given time windows further complicates the problem, making it hard to even find any solution for some specific instances, least of all an optimal one.

It is clear that brute force enumeration is not an option for this problem. Even the state-of-the-art solvers are not fast enough to compute a schedule within seconds (they actually take several minutes to solve rather simple in-

stances [23]). As mentioned before, the existence of time windows for the execution of the tasks makes it challenging to develop heuristic algorithms for quick computation of a solution. Also, though optimality may be out of reach for this problem, the quality of the solution cannot be completely disregarded.

Our approach to this problem is to break it down into sub-problems and develop a tailor-made algorithm to iterate through the sub-problems in order to find a solution to the original problem. Some of the sub-problems require optimization and are therefore solved by means of optimizing general purpose solvers. The choice of the solvers for the specific sub-problem (mainly between MILP and SMT) is also a result of this research, as different solvers have been tested over benchmark instances of well-known combinatorial optimization problems.

1.4 Research Questions

This thesis explores the following research questions:

RQ1 What are the strengths and weaknesses of SMT solvers and how do they compare to MILP solvers when used to solve industrial problems?

SMT solvers emerged within the computer science community and were originally employed for software and hardware verification. More recently they have been used as solving tools in other fields. Therefore evaluating them on different industrial problems could reveal their strengths and weaknesses. Moreover, it is interesting to evaluate SMT solvers against MILP solvers on different classes of problems and find out whether one technology outperforms the other, when and, possibly, why.

RQ2 How can the strengths of SMT and/or MILP solvers be exploited and combined to design an efficient algorithm for the CF-EVRP?

A strategy to solve the CF-EVRP in shorter time is to break it down into sub-problems. Since each sub-problem is solved on its own, different strategies can be applied depending on the problem features. MILP and/or SMT could be used as back-end solvers in a compositional algorithm designed to find a solution to the overall problem by iteratively solving the sub-problems. The method does not guarantee to find the optimum but it can still provide close-to-optimal solutions while significantly shortening the solving time.

1.5 Methods

In order to gather information about MILP and SMT, different problem classes have been studied: Job Shop (JSP) [24], Vehicle Routing (VRP) [25], and Bin Sorting (BSP) [26]. For each of these problems, benchmark instances are available (see Section 2.2). In this thesis, mathematical models have been formulated, based on existing literature about the problems and then solved using both MILP and SMT solvers. Comparison is based on running time necessary to find the optimum, or accuracy (gap between optimum and current best estimate) when timeout is reached.

For the CF-EVRP, we formulated a monolithic approach and evaluated it on some generated problem instances using SMT; We then broke down the problem into sub-problems and for each of them, we formulated mathematical models and designed a compositional algorithm to iteratively solve the sub-problems to find a solution to the CF-EVRP. Some sub-problems are solved using SMT solvers, while others are solved using MILP solvers.

1.6 Contributions

The contributions of this thesis are the following:

- Evaluation of different model formulations of three well-known optimization problems, namely the JSP, the BSP, and the VRP, implemented both as MILP and SMT;
- Formulation of the CF-EVRP problem;
- Decomposition of the CF-EVRP into sub-problems and mathematical formulation of each of them;
- Design of the compositional algorithm ComSat to find a solution to the CF-EVRP by iteratively solve the sub-problems it is decomposed into;
- Evaluation of the performance of ComSat against a monolithic formulation of the CF-EVRP implemented as SMT, over a set of generated benchmark instances;
- Improvements on ComSat by exploiting SMT *Unsat Core*.

1.7 Outline

This thesis consists of two parts. Part I is a general introduction to the field and puts the appended papers into context. Part II contains the appended papers. Part I is organized as follows: Chapter 2 gives an overview on the topic of problem complexity and provides examples through some specific problems investigated during this research project. Chapter 3 provides background knowledge about MILP and SMT, and compares these two methods over the problems presented in Chapter 2. Chapter 4 gives an in-depth overview of the CF-EVRP. Chapter 5 formally describes the CF-EVRP and presents the monolithic approach (MonoMod) and the compositional algorithm ComSat. Chapter 6 summarizes the contributions of the included papers. The thesis ends with closing remarks and directions for future work in Chapter 7.

CHAPTER 2

Optimization Problems in Automation

In the previous chapter we qualitatively introduced the concept of *problem complexity* and discussed the issue of dealing with hard problems, especially when time is a constraining factor and a solution cannot take too long to generate. In this chapter, problem complexity is discussed in more quantitative terms in order to be able to define the complexity of the CF-EVRP and motivate the need to design an algorithm that can handle such complexity and compute a solution in an acceptable time.

2.1 Problem Complexity and Efficient Algorithms

In the previous chapter, the words *problem* and *solution* have been used several times; let us now provide some formal definitions to better understand the following topics.

- *Variable*: a symbol that represents an element of the studied system and may assume any value from a predefined set of values called the *domain*;
- *Constraint*: a condition involving a subset of the problem's variables;

- *Objective function*: a function involving a subset of the problem's variables, where the domain is the cross product of the domain of the function's variables, and the co-domain is a real value to be maximized or minimized;

Given a real-world problem, it is possible to formulate a mathematical *model* by defining a set of variables to represent the different entities of the system of interest, and a set of constraints to represent how these different entities interact with each other. Moreover, if the real-world problem is an *optimization problem*, the model will also include an objective function. An assignment of values to the variables from their respective domains can be be divided into two categories:

- *Feasible*: if all constraints in the model are satisfied.
- *Infeasible*: if at least one constraint in the model is not satisfied.

Feasible assignments, henceforth called *solutions*, can be further divided into two categories:

- *Optimal*: if the assignment of values leads to the smallest (or largest) possible value of the objective function, given the valid domain of the variables, for a minimization (or maximization) problem. Note, that there can exist several optimal solutions to the same problem, i.e., different solutions with the same objective function value;
- *Sub-Optimal*: if the solution is worse than optimal.

Some additional definitions needed to talk about problem complexity:

- *Algorithm*: a finite sequence of instructions that can be mechanically carried out, such as a computer program that always terminates [27].
- *Search-space*: the set of all possible assignments to the problem, both feasible and infeasible.

The search-space is introduced in order to discuss the efficiency of different algorithms that aim to solve the same problem. In fact, an inefficient algorithm such as brute-force enumeration may have to search the entire search-space before terminating. On the other hand, when an efficient algorithm exists to

solve a problem, it may be sufficient to check only a few assignments before finding a solution.

It is now possible to classify problems based on the efficiency of the known algorithms to solve them. We say a problem is *tractable* if there exists an efficient algorithm to solve it, *hard* otherwise. In turn, an algorithm is said to be *efficient* if it runs in polynomial time, i.e., it can find a solution to a problem in a number of steps, and therefore a time, that is in the worst case polynomially proportional to the size of the problem [28]. If the number of steps is worse than polynomial, the algorithm cannot be called efficient.

Let us clarify this point by first talking about the size of a problem. For a given problem, there may exist instances of different sizes. The size is usually defined by one or more parameters of the problem. For example, let us assume that we want to sort a list of numbers in increasing order. What is the complexity of this problem? How large is its search-space? How efficient is an algorithm to solve the problem? We are going to formulate the problem as an assignment problem where the numbers to sort have to be assigned a position in the sorted list. The first thing to do is to understand what determines the problem size. In this case it is the number of elements to sort. The next step is to formulate a mathematical model of the problem. Let us define the set of variables that describe the problem. This will also allow us to determine the search-space.

Let $\mathcal{I} = \{\iota_1, \ldots, \iota_n\}$ be the set of n items to sort. Then let x_ι be the variable modelling the position of item $\iota \in \mathcal{I}$ in the sorted list. We will use one variable for each item in $\mathcal{I}$ and each variable can be assigned a value in the integer range $[0, n]$. Therefore the search-space size of this problem, using this modelling approach, and including both feasible and infeasible assignments, is n^n. To be more specific, the number of possible permutations of a list of elements of size n is $n! \leq n^n$; however, using the set of variables $x_\iota,\ \iota \in \mathcal{I}$ there can be (infeasible) assignments where two or more variables have the same value, meaning that two or more elements occupy the same position.

The next step is to formulate the constraints that relate the variables to each other. To do so, let us define, for each element ι in $\mathcal{I}$, the set of elements from $\mathcal{I}$ that are smaller than or equal to ι; let such set be $\mathcal{S}_\iota = \{\iota' \in \mathcal{I} \mid \iota' \leq \iota\},\ \forall \iota \in \mathcal{I}$. Note, that $\mathcal{S}_\iota = \varnothing$ when ι is the smallest element in the list.

Then we have the problem formulation:

$$1 \leq x_\iota \leq n, \qquad \forall \iota \in \mathcal{I} \tag{2.1}$$

$$x_\iota \neq x_{\iota'} \qquad \forall \iota, \iota' \in \mathcal{I},\ \iota \neq \iota' \tag{2.2}$$

$$x_\iota \geq x_{\iota'} \qquad \forall \iota \in \mathcal{I},\ \iota' \in \mathcal{S}_\iota \tag{2.3}$$

Constraint (2.1) defines the domain of the variables; constraint (2.2) forces each value to be placed in a different position; constraint (2.3) places larger values after smaller ones. By changing the sign in (2.3) from $\geq$ to $\leq$ the solution to the model will be a list sorted in descending order.

This formulation could be implemented and solved using a MILP solver; the best algorithms known for MILP are worse than polynomial in time [29]. However, sorting a list in ascending (or descending) order is known to be a simple problem. In fact, for a list of n numbers, it takes $n \log(n)$ steps in the worst case to sort them [30]. This example clarifies the point made in the previous chapter; reasoning about the problem allows to figure out its properties and exploit its structure to design better algorithms to solve it. Since *sorting* can be formulated as a MILP problem, it is possible, but not efficient, to use a MILP solver and compute a solution. However, *sorting* does possess a structure that allows to solve it in polynomial time by designing a specific algorithm for the purpose.

P vs. NP

In this section the definition of *tractable* and *hard* problems is formalized by introducing the complexity classes $\mathcal{P}$ and $\mathcal{NP}$. There exist several complexity classes, but for the current discussion we restrict ourselves to $\mathcal{P}$ and $\mathcal{NP}$.

Before talking about complexity classes though, it is necessary to define a class of problems called *decision problems*. A decision problem is a problem whose solution is a *yes/no* answer [31]. For instance, given a model, verifying whether there exists a feasible assignment or not, is a decision problem.

Also, the following discussion is based on the assumption that no efficient algorithm exists to solve problems in $\mathcal{NP}$. Such assumption is commonly believed to be true, though any attempt to prove it has failed so far.

$\mathcal{P}$ is the class of decision problems for which there exist a polynomial algorithm to solve them; a problem in $\mathcal{P}$ is called *tractable*. For some decision problems that do not belong to $\mathcal{P}$ it is still possible, given a solution to the

problem, to verify its correctness in polynomial time. These problems belong to the class $\mathcal{NP}$. Of course, if solving a problem takes polynomial time, so does verifying a given solution, therefore $\mathcal{P} \subseteq \mathcal{NP}$. Problems in $\mathcal{NP}$ that are not in $\mathcal{P}$ are said to be *hard*.

Moreover, it is possible to classify the relative complexity of problems by reducing (transforming) them into other problems whose complexity is known. In order to formalize this concept, let us define *polynomial-time reduction*.

Polynomial-time reduction is the transformation of one problem into another in polynomial time. If it is possible to reduce or transform a problem X into another problem Y in a number of steps that is polynomial to the size of problem X, then X is *polynomial-time reducible* to Y. In other words, we can claim that "X is at least as hard as Y". In fact, assuming that there exist a *black box* capable of solving Y in polynomial time, then X is solvable in polynomial time too. For more details about polynomial reduction see [28].

A decision problem C is said to be $\mathcal{NP}$*-complete* if it belongs to $\mathcal{NP}$ and all problems in $\mathcal{NP}$ have polynomial-time reduction to C. Over the decades, many problems have been proven to be $\mathcal{NP}$-complete [28].

Everything that has been said so far about $\mathcal{P}$ and $\mathcal{NP}$ holds as long as no efficient algorithm is found to solve a problem in $\mathcal{NP}$. If that happens, because of the polynomial-time reduction, all $\mathcal{NP}$-complete problems could be efficiently solved, and $\mathcal{P} = \mathcal{NP}$.

So far, we provided definitions that apply to decision problems, although in this thesis, we actually focus on finding solutions, not only to verify whether existing solutions are valid. However, according to [32], for $\mathcal{NP}$-complete problems, finding the actual solution is not harder than finding out whether one exists or not.

When talking about optimization problems though, the same reasoning does not apply. Given a solution to an optimization problem, it is not guaranteed that there is an efficient way to verify that it is the optimal solution. Optimization problems belong to the class of $\mathcal{NP}$-hard problems, i.e., the class of problems that are at least as hard as the $\mathcal{NP}$-complete problems, and not necessarily in $\mathcal{NP}$.

This section only briefly introduces the main ideas behind complexity classes, for further details we refer the reader to [33]. However, the brief introduction provided can help to understand the need of figuring out more efficient meth-

ods to deal with *hard* problems such as the CF-EVRP. In the next section are presented some known $\mathcal{NP}$-hard, optimization problems, investigated to evaluate the performance of SMT and MILP solvers and used to design an algorithm for the CF-EVRP (see Chapter 5).

2.2 Complexity of Known Optimization Problems

Let us now continue our discussion on complexity by introducing the three problems that have been used as benchmarks for comparing general purpose solvers in the first half of this research project; for more details we reference the reader to the appended papers.

The Job Shop Problem

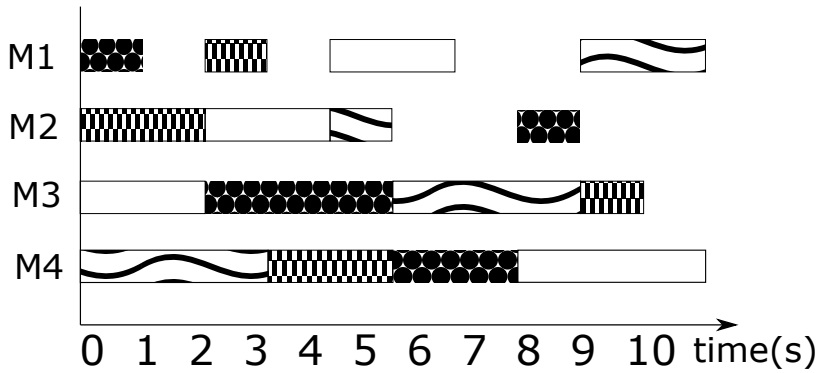

Figure 2.1: Illustration of a possible solution to a job shop problem.

The *job shop problem* (JSP) [34] is the $\mathcal{NP}$-hard [35] optimization problem of assigning machines to jobs in such a fashion that the make-span is minimized. In the JSP each job has to visit each machine in order to be finished; the order of the visits is pre-defined. Also, machines can only execute one job at a time; the duration of a job's visit on a machine is fixed and given as input. The problem has been studied for decades and there exists a vast literature on exact and approximate methods to solve it [36], together with a large set of benchmark problems ranging from small to very large. Figure 2.1 shows a possible schedule for a problem concerning four jobs and four

machines. The different patterns in the rectangles represent different jobs; the rectangles length represent the duration of the job's visit on the machine. There exists plenty of literature and different sets of benchmark problems on the VRP available for comparison [34], [37]–[40].

The JSP has variants featuring additional requirements; the flexible job shop problem (FJSP) [41], where operations can be executed by more than one machine; the *no-buffer* JSP [42] where jobs cannot leave a machine until the next machine in the sequence is available for processing, since there are no buffers to hold the parts. A sub-variant of this problem involves limited-capacity buffers; there is the *no-wait* JSP, which involves constraints on the elapsing time between operations of the same job, due to the perishability of the goods, also described in [42]. Further details about JSPs and previous studies on the subject are presented in Paper A.

For the standard JSP, even though finding the optimal solution, or even proving that a solution is optimal is hard, it is possible to design heuristic algorithms to produce sub-optimal solution very quickly. One would be to simply execute all operations sequentially. It would yield a very bad objective function value, but it would be a feasible assignment, and it would take virtually no time to compute.

The Bin Sorting Problem

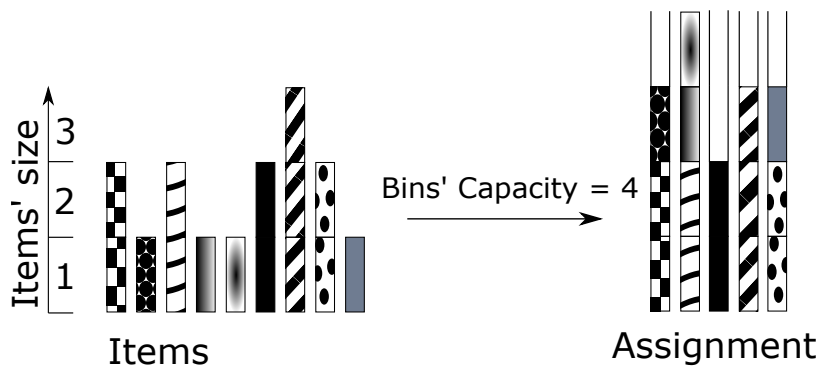

Figure 2.2: Illustration of a possible assignment for a bin sorting problem.

The bin sorting problem (BSP) is the $\mathcal{NP}$-hard [43] optimization problem

of fitting items into bins such that the number of bins is optimal. Items are characterized by a value indicating their size (in real world scenarios this could represent their weight or their volume) and there exist two versions of the BSP, one being the dual of the other. In the bin packing problem (BPP) [26], there is a maximum capacity of the bins that cannot be exceeded and the goal is to minimize the number of bins; in the bin covering problem (BCP), there is a minimum capacity of the bins that cannot be under-reached and the goal is to maximize the number of bins. Figure 2.2 shows a BPP where nine items have to be packed in bins of maximum capacity *four*.

Especially for the BPP there is plenty of literature available for comparison [44], [45], as well as five different sets of standard benchmark problems [46]–[50]. The literature for the BCP is less exhaustive and slightly outdated, possibly because the problem is so closely related to the BPP [51]; However, more recent work on online BCP is presented in [52]. Further details about VRPs and previous studies on the subject are presented in Paper B.

Also for the bin sorting problem, it would be possible to design heuristic algorithms to provide sub-optimal solutions quickly. For the BPP for instance, the simplest solution (and the worst one) would be to pack an item in each bin; a better one would be to pack items in a bin until no additional item fits and then repeat the algorithm with more bins until all items are packed.

The Vehicle Routing Problem

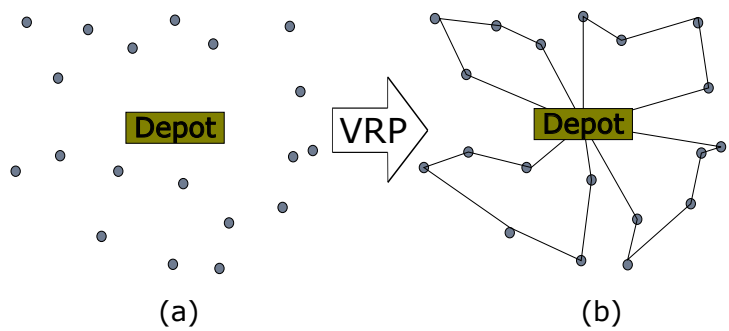

Figure 2.3: Illustration of the vehicle routing problem.

The Vehicle Routing Problem (VRP) [53] is an $\mathcal{NP}$-hard [54] optimization

problem of computing routes to serve customers while minimizing a cost function, typically the travelled distance, or the number of vehicles required (or a combination of both). Customers are also characterized by a *service time*, which is the time the vehicle serving the customer has to spend at the customer's location in order to serve it. In the VRP, for a route to be valid, it has to start and end at the same depot (in case of multiple depots). Moreover, each customer has to be visited by exactly one vehicle. There exist many different extensions of the basic problem, involving additional constraints such as limited capacity of goods that a vehicle can carry (this corresponds to specific demands of goods for each customer) [55], limited operating range of the vehicles in terms of travelled distance [56], time windows to deliver goods (i.e. earliest and latest arrival time at a customer) [57], multiple depot stations, etc. A common trait when dealing with the VRP is to treat the real world map as a graph, where each point of interest (i.e. depots, customers) is a node and two nodes are connected with each other by a weighted edge representing their distance. There exists plenty of literature and different sets of benchmark problems on the VRP available for comparison [56], [58]. Also, [59] presents a review of different models, classifications and solving algorithms. Further details about VRPs and previous studies on the subject are presented in papers C, D, and E.

Once again, finding the optimal solution is hard, but, as for the previous two problems, it is possible to develop heuristic solutions that will produce sub-optimal solutions quickly. For instance, having one vehicle serving all the customers, or having one vehicle for each customer.

2.3 Complexity of the CF-EVRP

So far we have seen problems that are hard to solve to optimality, but tractable if the goal is to find a sub-optimal solution for them. As we mentioned in the previous chapter though, there exist problems that are hard even when the goal is finding any solution at all. In scheduling, one factor that seems to further complicate things is the existence of time windows for the execution of an operation (JSP) or the serving of a customer (VRP). In fact there is no guarantee that the solutions provided by an heuristic algorithm will meet the time windows for the operations/customers.

This is exactly what happens in the CF-EVRP. The time windows for the

delivery of goods to the workstations constrain the execution time, while the limited capacity of the road segments may require the vehicles to wait for each other or to take longer routes in order to avoid blocking each other's way. In such a scenario it is impossible to define a simple heuristic to find a solution, because infeasibility can be caused by a combination of factors. It is therefore necessary to try different assignments and backtrack when one turns out to be infeasible.

The approach described in the previous paragraph may sound naive, since it is basically a *trial and error* strategy. However, there is still much reasoning that can be done in order to avoid trying assignments that are obviously infeasible. Moreover, it is also possible to *learn* from infeasible assignments and steer the search based on them. In Chapter 5 is presented an in-depth discussion over the Compositional Algorithm (ComSat) developed to solve the CF-EVRP by efficiently and incrementally build an overall solution based on the solutions of the sub-problems the CF-EVRP is broken into.

Before diving into the details of how to solve the CF-EVRP efficiently, an overview on MILP and SMT is given in the next chapter, and then a formal description of CF-EVRP itself is given in Chapter 4.

CHAPTER 3

On Satisfiability Modulo Theory and Mixed Integer Linear Programming

As MILP and SMT play an important role in this thesis, we will here give a brief introduction on how to utilize these techniques. The purpose of the following sections is to present the main concepts regarding the two approaches; therefore no proofs are provided nor any in-depth explanations. Instead, we refer the reader to relevant literature. For a thorough introduction to MILP see [60], and for SMT see, for example, [61].

3.1 Mixed Integer Linear Programming (MILP)

In order to understand how MILP works, it is necessary to first understand *linear programming* (LP). The difference between MILP and LP is that in LP all variables are real-valued, while in MILP a subset of the variables can be integer, or binary. An LP problem is a conjunction of linear inequalities together with a linear objective function over a set of real variables. As a convention, a linear problem is a minimization, though it can easily be converted into maximization (or the other way around) by changing the sign of

the objective function.

A general linear problem counting n variables and m constraints takes the *standard* form:

$$\begin{aligned} \min\ & c^T \vec{x} \\ \text{such that } & A_{m,n}\vec{x} \leq b \\ & \vec{x} \geq 0 \end{aligned}$$

where $\vec{x} = [x_1, \ldots, x_n]$ is a vector containing the decision variables, $c^T = [c_1, \ldots, c_n]$ is an array of coefficients for the objective function, b is a column vector of the right side values, and $A_{m,n}$ is a coefficient matrix.

The following example with two variables x and y can be shown on the plane where each constraint is represented by a line and since the problem is a conjunction of them, they all have to be fulfilled, together with the $\vec{x} \geq 0$ constraint, thus defining an area called the *feasible region*, a portion of the plane where all possible solutions to the problem lie.

$$\begin{aligned} \min\ & -0.75x - y && (3.1) \\ \text{such that } & x + y \leq 4 && (3.2) \\ & 1.5x + y \leq 5 && (3.3) \\ & x, y \geq 0 && (3.4) \end{aligned}$$

In the problem, (3.1) is the objective function, (3.2) and (3.3) are the linear inequalities, and (3.4) is the non-negativity constraint over the variables.

In Figure 3.1 (a), constraints (3.2) and (3.3), represented by the blue and red line, together with the non-negativity of the variables, define the feasible region shown as the gray area. Any feasible solution to the problem lies within that area, while any point in the white area conflicts with at least one constraint. By formulating a problem in standard form, i.e., by having only linear inequalities with the symbol "$\leq$", it is possible to represent it graphically (as shown in Figure 3.1). It is always possible, given a linear problem, to convert it into standard form. When all constraints are linear, the feasible region has the shape of a convex polygon and the optimal solution corresponds to one or more of its extreme points.

The objective function represented by the green line in Figure 3.1 (b) is

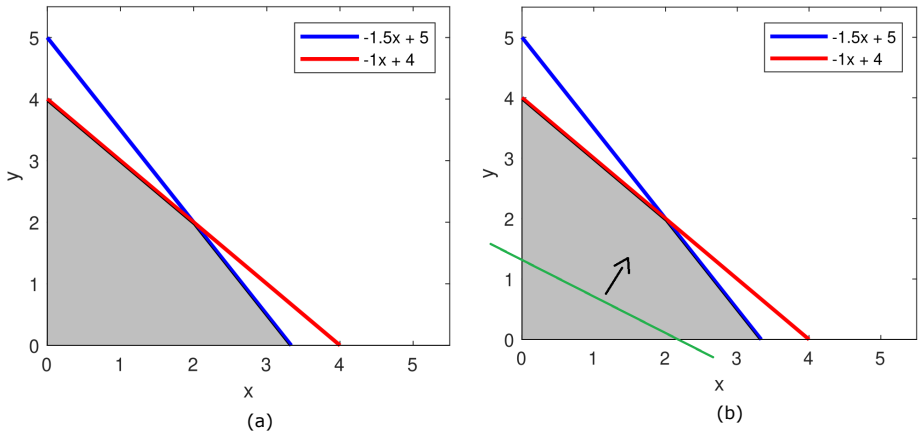

Figure 3.1: Illustration of the feasible region (a) and objective function (b) for the LP example.

used to determine which extreme point(s) corresponds to the optimal solution. In fact, let's assume it is possible to shift the objective function along its perpendicular and position it on the extreme points of the polygon. Depending on its expression, its value will increase or decrease as it shifts upwards or downwards. In the example, since the coefficients of x and y are negative, when shifting it upwards its value will decrease. When we position it on an extreme point, the corresponding solution is given by solving the system of inequalities that represent the lines touching that point. At the same time the objective function value is calculated by plugging in the solution into the objective function expression.

In the example above, the extreme points of the feasible region are

$$(0,0), (10/3, 0), (2,2), (0,4),$$

the objective function can assume the values $0, -2.5, -3.5, -4$ when touching each point. Since the problem is a minimization, the optimal solution is the one corresponding to the smallest value of the objective function, i.e., the point $(0,4)$. Graphically, we can see as $(0,4)$ is the furthest point that the objective function can touch before "leaving" the feasible region.

One particular case is when the objective function is parallel to one of the

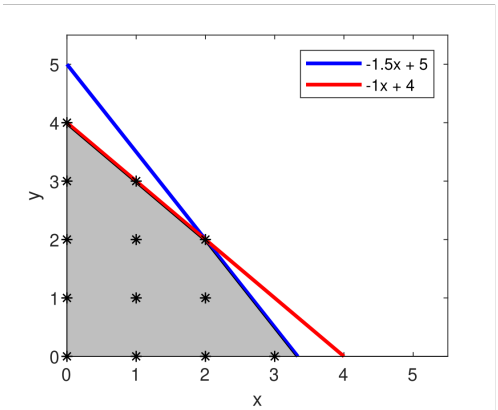

Figure 3.2: Illustration of the feasible region when the integrality constraint applies.

lines touching the extreme point to the optimal solution. In that case, in fact, the objective function will be touching two points, not one. Both points are optimal, as well as all the points in between them. In this case there is an infinite number of optimal solutions. It could also be thought that there exist no feasible region, i.e., no assignment to the variables that satisfies all the constraints.

As of today, the most widely used algorithm to solve linear problems is the Simplex algorithm [62]. Its complexity is in the worse case exponential in the problem size; nonetheless it performs pretty well in practice. There is a polynomial algorithm to solve linear problems, the *Interior Points* algorithm [63], that in practice though does not perform as well as the Simplex algorithm.

When the variables are restricted to be integer, there is no guarantee that the optimum lies on a vertex. Figure 3.2 shows again the feasible region of a problem but this time the integrality constraint applies and only the points marked by a star are actually feasible. In this case, the extreme point corresponding to the optimal solution, happens to be integer, therefore solving the relaxed LP would yield a feasible solution to the MILP problem too.

One possibility to solve a mixed integer linear problem is to apply a branch and bound algorithm; based on the fact that solving a relaxed version of the problem,i.e. the integrality constraint on all the variables is removed, yields a solution at least as good or better than the solution of the original problem,

the algorithm iteratively solves relaxed problems to find tighter and tighter bounds for the original one, until the gap between the bounds falls below a desired threshold.

An example can be used to clarify the procedure:

$$\begin{aligned} \min \ & -50x_1 - 20x_2 - 60x_3 \\ s.t. \ & 2x_1 + x_2 + 2x_3 \leq 120.5 \\ & 3x_1 + 2x_2 + 2x_3 \leq 150 \\ & x_1 \geq 20 \\ & x_1, x_2, x_3 \in \mathbb{N} \end{aligned}$$

In the problem above, the integrality constraint over the variables is included; this means that the objective function value z may be fractional, depending on the coefficients (this is not the case here since they are all integer as well), but the values of the decision variables x_1, x_2 and x_3 must be integer.

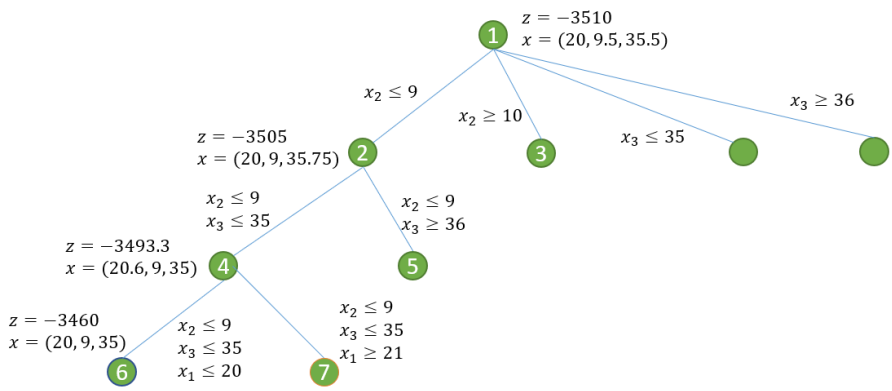

Figure 3.3: Illustration of branch and bound steps.

Figure 3.3 shows some of the steps that a branch and bound algorithm would take to find an (possible multiple equally good) optimal solution to the problem. In step 1 the relaxed problem is solved and the optimal value is 3510 but the solution is not feasible for the original problem because x_2 and x_3 are not integer numbers. This leads to four alternatives, represented by nodes

2, 3, 8, and 9, respectively. An additional inequality is added that restricts the variable domain to be either smaller than or equal to the floor or larger than or equal to the ceiling of its current value. In node 2, x_2 is restricted to be smaller than or equal to 9 (its value from the previous iteration was 9.5). The relaxed problem with the additional constraint is solved again and this time the optimum is 3480 but the solution is still unfeasible for the original problem, since x_3 is 35.75. We can branch again by adding another inequality: either x_3 is larger than or equal to 36 (ceiling of 35.75) or it is smaller than or equal to 35 (floor of 35.75).

The process goes on until a feasible integer solution is found, as happens in node 6. Now we have a lower bound for the original problem $\underline{z} = 3460$. From now on, every time we explore a branch we can stop whenever we found a solution whose value is smaller than the current $\underline{z}$ and prune (stop exploring) that branch because we know that no better solution can be found there since going down a branch we can only find worse and worse solutions because we keep shrinking the feasible region.

When an integer solution is found, the search on that branch is over. If the optimal value for the current relaxed problem is better than the current lower bound, this becomes the new lower bound and the search starts on another branch. The algorithm terminates when all branches are either searched or pruned. Note that this example concerns the maximization of the objective function.

There exist techniques that build on top of the branch and bound algorithm to increase the performance but as of today, solving a mixed integer linear problem is computationally demanding and many problems still remain intractable [64]. The reader is referred to [60] for further details.

3.2 Satisfiability Modulo Theories (SMT)

In this section a background on Satisfiability Modulo Theories (SMT) is given; the focus is on describing how SMT solvers find satisfiable assignments. The search for optimal solutions is not described in detail as there are many different algorithms that can be built on top of a solver to drive its search for the optimum, while the decision process for feasibility is common among them all. However, a brief discussion on optimizing SMT solvers is provided at the end of the section. In the following, some terminology is given:

Literal: a literal is either a variable or its negation. A literal is negative if it is a negated variable, and positive otherwise. A positive literal is *satisfied* if its variable is assigned to *True*. Similarly, a negative literal is *satisfied* if its variable is assigned to *False*.

Clause: several literals connected by logical disjunction $\vee$ [65]. A collection of one or more conjuncted and/or disjuncted clauses is called a *formula*. An *empty* clause is always *False*.

Conjunctive Normal Form (CNF): a formula is in conjunctive normal form if it is a conjunction of disjunctions of literals, i.e., it has the form

$$\bigwedge_i \left(\bigvee_j l_{ij} \right),$$

where l_{ij} is the j-th literal of the i-th clause. An *empty* CNF formula is always *True*.

In order to understand how SMT solvers work, it is necessary to discuss Boolean satisfiability [66] (SAT). A SAT problem is formed by a formula of Boolean variables in conjunctive normal form. Solving a SAT problem means finding an assignment to each variable (either *True* or *False*) that makes the whole formula *True*, or providing a counterexample that shows that the formula is unfeasible. Although SAT is in general $\mathcal{NP}$-complete [67], over the years SAT solvers have become very efficient and can nowadays solve large problems counting even millions of variables and clauses in relatively short time [68].

When modelling a system, often CNFs are not the most user-friendly way to represent its behaviour; fortunately, there exist ways to transform any formula into a CNF formula, and it can be done efficiently by using Tseitin's linear encoding [69]. The reason why it is worth to spend computation time on transforming a formula into CNF is that since it is a conjunction of clauses, as soon as one clause is determined unsatisfiable, the whole formula is unsatisfiable. This property can be exploited by SAT solvers to efficiently solve many problems involving a high number of variables.

State of a clause under an assignment: a clause is **satisfied** if one or more its its literals are satisfied (*True*), **conflicting** if all of its literals are assigned but the clause is not satisfied, **unit** if it is not satisfied and all but one of its literals are assigned, and **unresolved** otherwise.

If a clause is *unit*, this means that all but one of its literals are assigned but the clause is still not *satisfied*, i.e., all assigned literals are *False*. This means that the remaining unassigned literal has to be *True*, otherwise the clause would be *conflicting*. Therefore the remaining literal is **implied** by the clause.

The solver starts assigning literals based on some heuristic strategy, until one or more clauses become *unit* and *implies* one (or more) literals. This implication may turn more clauses into *units*, otherwise there is going to be more heuristic based assignments. The process goes on until all literals are assigned or a conflict arises because the implications made a clause *conflicting*.

Modern SAT solvers exploit the above mentioned properties of formulas in CNF by applying the *conflict driven clause learning* (CDCL) framework to the problem. The search-space can be thought of as a binary tree, in which nodes are partial assignments and leaves are full assignments. The solver traverses the tree and when a conflict is found, it "learns" from the conflict by adding a new clause to the model containing the information about the assignment that led to the conflict, backtracks to the point where the assignment was made and tries a different assignment. The newly added clause will prevent making the same (conflicting) assignment again. The solver keeps traversing and backtracking until it finds a feasible assignment or no assignment exists that satisfies all the clauses, including the *learned* ones.

Example of the CDCL procedure

Let us clarify the concepts introduced so far with a small example. Assume we want to find an assignment for the following formula:

$$(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_3 \vee x_1). \tag{3.5}$$

Initially, no literal is implied so the choice to satisfy one literal rather than another is merely dependent on the heuristic strategy used. Remember that, since the formula is a conjunction of clauses, as soon as one clause is conflicting, the whole formula is not satisfied by that assignment.

Let us assign the value *False* to the variable x_1 in order to satisfy the literal $\neg x_1$ in the first clause. The second clause would then imply the variable x_2 to be *True*, since its first literal x_1 is not satisfied. Now x_3 is implied by the third clause to be *False*, since the first literal in it, $\neg x_2$ is not satisfied. Finally, the last implication makes the fourth clause conflicting, because it forces the variable x_1 to be *True*, since the first literal in it is not satisfied, but we previously assigned to x_1 the value *False*.

Thus it is necessary to backtrack where the decision that eventually led to the conflict was made and make a different one. In this case, that decision was assigning the value *False* to x_1.

In order to avoid repeating the same mistake, we will add a clause saying that x_1 must be *True*, to the formula, i.e., $\neg(\neg x_1)$. In this case the new *learned* clause only contains one literal, which forces the literal to be satisfied. Satisfying x_1 also satisfies the second and fourth clause and makes x_2 implied by clause *one*, which in turn makes $\neg x_3$ implied by clause *three*.

We have now a full assignment that satisfies (3.5):

$$x_1 = x_2 = \mathit{True},\ x_3 = \mathit{False}$$

From SAT to SMT

While SAT solvers can be extremely efficient in dealing with large models, Boolean satisfiability is not expressive enough to easily model many real-world industrial problems. Boolean variables and propositional logic can be handy when it comes to model binary decision (executing this operation or not), but integer and real variables and linear arithmetic are necessary to describe

other important features (operations' duration for instance). As it happens, SMT solvers are able to use integer and real variables over a range of different theories, including combinations of theories. SMT solvers can do so in two different ways: the **eager approach** or the **lazy approach** [70].

However, before going into details about the eager and lazy approaches, it is necessary to provide some more terminology:

- *logical symbols*: standard Boolean connectives (e.g. "$\wedge$","$\neg$"), quantifiers ("$\exists$" and "$\forall$"), and parenthesis;
- *non-logical symbols*: functions, predicates, and constant symbols. A set of non-logical symbols is called a *signature* and is denoted by the symbol Σ;
- $\Sigma-$*formula*: formula that uses only the non-logical symbols from Σ (possibly in addition to logical symbols, this is why the distinction between logical and non-logical symbols is needed);
- *free variable*: variable that is not bound by a quantifier, i.e., $\exists$ or $\forall$;
- *sentence*: formula without free variables;
- *syntax*: rules for constructing formulas.

The distinction between *logical* and *non-logical* symbols is necessary because, while the clause of a theory may or may not be constructed using logical symbols, clauses are combined using logical symbols, regardless of the theory they belong to. For instance, a formula in linear arithmetic, is a conjunction of clauses, where each clause is a linear equality/inequality.

In a theory, the syntax is needed to *interpret* the non-logical symbols. For instance, the symbol "+" is usually associated to addition, but this may not be true for some theories. Hence, we need rules to use the non-logical symbols to construct the formulas.

Theory T: a theory is a set of Σ-sentences. For a given Σ-theory, a Σ-formula φ is **T-satisfiable** if there exists an assignment that satisfies both the formula and the sentences of T.

In other words, we use a set of *sentences* to define the syntax of a theory, i.e. to define the interpretation of the non-logical symbols.

Eager Approach

One possibility to solve a problem involving a combination of theories is to convert its model into SAT. For each variable, depending on its domain, a number of Boolean variables are generated and for each constraint, a set of clauses. Then the assignment problem is solved using the CDCL framework as described above.

This procedure is called *bit blasting* and can be computationally expensive. For each variable in the original problem there will be as many Boolean variables as the size of the original variable domain. When it comes to constraints translation, the procedure can be even more expensive. Let us clarify this point with an example; the goal is to find an assignment to x_1 and x_2 such that x_1 is smaller than or equal to x_2, and their domains can be either 0, 1, or 2.

$$x_1 \leq x_2 \tag{3.6}$$

$$x_1, x_2 \in \{0, 1, 2\} \tag{3.7}$$

To turn this problem into a SAT formulation, we define the following Boolean variables:

$$x_{ij} \quad i \in \{0, 1\},\ j \in \{0, 1, 2\}.$$

There is one Boolean variable for each variable in the original problem and for each element in the variable domain.

The linear inequality in 3.6 can be converted as follows:

$$x_{ij} \implies \bigwedge_{\substack{j' \in \{0,1,2\} \\ j' \neq j}} \neg x_{ij'} \qquad \forall i \in \{0, 1\},\ j \in \{0, 1, 2\} \tag{3.8}$$

$$x_{1j} \implies \bigwedge_{\substack{j' \in \{0,1,2\} \\ j' < j}} \neg x_{2j'} \qquad \forall j \in \{0, 1, 2\} \tag{3.9}$$

(3.8) states that if one of the Boolean variables representing each original variable is *True*, all the other Boolean variables representing the same original variable must be *False*. This is equivalent to say that *exactly one* of the Boolean variables representing each original variable can be *True* and this constraint is necessary because each original variable can assume only one

value from its domain; (3.9) states that if the Boolean variable representing x_1 having the value j is *True*, none of the variables representing x_2 having a value strictly smaller than j can be true.

While the eager approach can successfully be applied to finite domain theories, it is not clear how it would be possible to reduce linear real arithmetic literals to a Boolean satisfiability problem and since in this thesis we deal with finite domain theories, the subject is not further investigated, though the reader is referred to [71] for further details.

Lazy Approach

The lazy approach combines the underlying SAT solver with a theorem prover to decide the assignment of variables. This means that the SMT solver must be able to recognize the theory it is dealing with and must be equipped with a prover appropriate for that theory.

A formula of a certain theory is a logical combination of clauses belonging to that theory. The solver will replace each clause in the formula with a Boolean variable and call the SAT solver to find a satisfiable assignment. If such assignment exists, the solver will call the theorem prover to check whether the current assignment is feasible within the theory. If not, a new clause is learned and the process starts again.

Once again, an example can aid understanding. Consider the formula:

$$(x_1 = x_3 \vee x_1 = x_2) \wedge (x_1 = x_2 \vee x_1 = x_4) \wedge x_1 = x_2 \wedge x_1 \neq x_3 \wedge x_1 \neq x_4 \quad (3.10)$$

The theory involved in this formula is called *equality theory* [61] where literals are either equalities or inequalities. Given a conjunction of literals (that would be the outcome of the SAT solver), there exist procedures to check whether the assignment is satisfiable or not.

In this case, the SMT solver would generate Boolean variables:

$$c_1 : x_1 = x_2$$
$$c_2 : x_1 = x_3$$
$$c_3 : x_1 = x_4$$

c_i (for $i \in \{1, 2, 3\}$) will evaluate to *True* if the corresponding equality literal is indeed an equality, *False* otherwise.

Call the SAT solver to find an assignment for

$$(c_2 \vee c_1) \wedge (c_1 \vee c_3) \wedge c_1 \wedge \neg c_2 \wedge \neg c_3$$

Let us assume the solver finds the satisfiable assignment $c_1 \wedge \neg c_2 \wedge \neg c_3$; this means that the following is true: $x_1 = x_2 \wedge x_1 \neq x_3 \wedge x_1 \neq x_4$.

The solver will now call the theorem prover to check the assignment (which, in this case is satisfiable)

The examples presented in this section are inspired by [61], which is recommended reading for further details on techniques for solving SMT-problems.

Optimization Modulo Theory

As mentioned in Chapter 1, it is possible to express many practical industrial problems in quantitative terms and define a cost function to optimize. This led to efforts of supporting optimization in the SMT-based tools. In [72] an SMT variant is introduced where the theory involved in the problem becomes progressively *stronger*, meaning that more sentences are added to it as the optimization process takes place; this is done by implementing a branch-and-bound setting where the solver knows the cost function and the current best bound. Each time a better bound is found, models with a cost higher than this new bound become inconsistent with the theory. In [73] and [74], the authors describe some of the optimizing algorithms running under the hood of the SMT solver Z3, such as a collection of *MaxSAT* solvers, solvers that drive the search for an optimal solution by trying to satisfy as many *soft* constraints as possible, and a module for optimization of linear arithmetic objective functions; among others, the module contains a Simplex-based algorithm for the CDCL framework [75]. In [76], the authors provide insight on how optimization is achieved with the SMT solver OptiMathSAT. It is explained that unlike other optimizing SMT solvers, in which the solver is used as a black-box and the optimization proceeds through a sequence of incremental SMT calls, with OptiMathSAT the whole optimization procedure is pushed inside the SMT solver.

In conclusion, optimization with SMT solvers can be achieved in different ways and it is hard to tell which one is the most efficient, since different approaches perform differently depending on the problem.

The Unsat Core

Besides evaluating whether a set of constraints is feasible, it can be of interest to understand which constraints are the ones that cause infeasibility, if the formula is infeasible.

In general, a formula in CNF is infeasible if it is possible to generate an empty clause $\perp$ by *resolution* from the original clauses [77]. Two clauses can be *resolved* to generate a *resolvent clause* as long as there is one and only one variable that appears in both clauses, negated in one and non-negated in the other, The resolvent clause takes the disjunction of the remaining literals in both clauses.

For example, let the formula φ be the conjunction of

$$\begin{aligned} c_1 &= x_1 \vee x_2 \\ c_2 &= \neg x_2 \vee x_3 \\ c_3 &= \neg x_1 \\ c_4 &= \neg x_3, \end{aligned}$$

that is,

$$\varphi = (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \neg x_1 \wedge \neg x_3.$$

Then, resolving c_1 and c_2 is possible, since they only have x_2 in common, and x_2 is negated in c_1 and non-negated in c_2. Resolving c_1 and c_2 leads to the resolvent clause $c_5 = x_1 \vee x_3$. After this resolution, the formula becomes:

$$\varphi = (x_1 \vee x_3) \wedge \neg x_1 \wedge \neg x_3$$

Similarly, resolving c_5 and c_3 leads to the resolvent clause $c_6 = x_3$ since x_1 appears in both c_5 and c_3 with different sign, and $c_3 = \neg x_1 = \neg x_1 \vee \mathit{False}$. After this resolution, the formula becomes:

$$\varphi = x_3 \wedge \neg x_3$$

Finally, resolving c_6 and c_4 leads to $c_7 = \perp$. Since it is possible to generate an empty clause from the original clauses, φ is infeasible.

Unsatisfiable Core (or *Unsat Core*)**:** An *Unsat Core* is a set of clauses that make any assignment infeasible. More formally, given a set of clauses φ, $\mathcal{C}_i$

is an Unsat-Core of φ if $\mathcal{C}_i \subseteq \varphi$ and $\mathcal{C}_i$ is infeasible [77]. This means that a problem may have more than one *Unsat Core*. Moreover, *Unsat Cores* can have different cardinalities and one *Unsat Core* can be a subset of another *Unsat Core*.

***Minimal Unsat Core*:** An *Unsat Core* is said to be *minimal* if by removing any clause from it, the remaining clauses become feasible. Note, that a problem can have multiple minimal *Unsat Cores*.

In order to illustrate the definition of *Unsat Cores* and minimal *Unsat Cores*, a Boolean example from [77] will be treated. The example concerns a set of variables $x = \{x_1,\ x_2,\ x_3\}$, and a set of clauses $\varphi = \{\omega_1,\ \omega_2,\ \omega_3,\ \omega_4,\ \omega_5,\ \omega_6\}$ in CNF. The clauses are defined as:

$$\begin{array}{lll} \omega_1 = x_1 \vee \neg x_3 & \omega_2 = x_2 & \omega_3 = \neg x_2 \vee x_3 \\ \omega_4 = \neg x_2 \vee \neg x_3 & \omega_5 = x_2 \vee x_3 & \omega_6 = \neg x_1 \vee x_2 \vee \neg x_3 \end{array}$$

As an example, consider clauses ω_2, ω_3 and ω_4. For the clause ω_2 to be *True*, the variable x_2 must be *True*. Then, ω_3 implies that x_3 must be *True*, which in turn means that the clause ω_4 is *False*. Thus, this example is infeasible.

It turns out that this example has 9 different *Unsat Cores*, namely:

$$\begin{array}{ll} \mathcal{C}_1 = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6\} & \mathcal{C}_2 = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5\} \\ \mathcal{C}_3 = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_6\} & \mathcal{C}_4 = \{\omega_1, \omega_3, \omega_4, \omega_5, \omega_6\} \\ \mathcal{C}_5 = \{\omega_2, \omega_3, \omega_4, \omega_5, \omega_6\} & \mathcal{C}_6 = \{\omega_1, \omega_2, \omega_3, \omega_4\} \\ \mathcal{C}_7 = \{\omega_2, \omega_3, \omega_4, \omega_5\} & \mathcal{C}_8 = \{\omega_2, \omega_3, \omega_4, \omega_6\} \\ \mathcal{C}_9 = \{\omega_2, \omega_3, \omega_4\} & \end{array}$$

In Figure 3.4, it is shown how, when considering the inclusion operation, the set of *Unsat Cores* forms a sub-lattice [78], where the unique upper bound is always the entire set of clauses $\mathcal{C}_1$, while the lower bounds are the minimal *Unsat Cores*, namely $\mathcal{C}_4$ and $\mathcal{C}_9$.

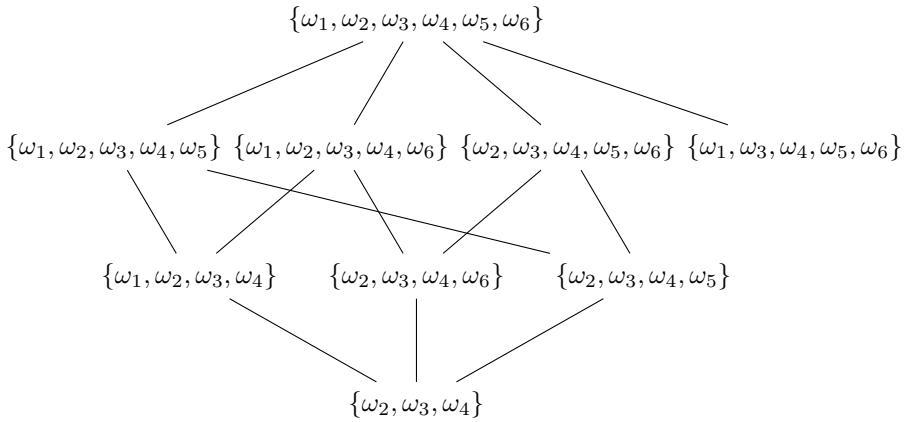

Figure 3.4: Lattice of the *Unsat Core* set.

Evaluating Infeasibility Scenarios

We now present four different scenarios where the formula is infeasible and one or more *Unsat Cores* can be extracted.

Scenario 1

In the first scenario, three constraints lead to infeasibility:

$$\begin{aligned} \omega_1 &: \; x + y < 0 \\ \omega_2 &: \; x = 1 \\ \omega_3 &: \; y = 1 \end{aligned}$$

If both x and y must be equal to 1, it is impossible for $x + y$ to be smaller than 0. When considered pairwise, these constraints do not generate any inconsistency. However, when considering all three of them at the same time, they are infeasible. Therefore, if any one of the constraints is removed, the entire problem becomes feasible. As an example, if constraint ω_2 is removed, any assignment where $x < -1$ would be a solution.

$$\{\omega_1, \omega_2, \omega_3\}$$

Figure 3.5: Lattice of the *Unsat Core* set of Scenario 1.

As shown in Figure 3.5, the *Unsat Core* set only contains one element, including all three constraints.

Scenario 2

In the second scenario there are three constraints, each of them being infeasible with either of the other ones:

$$\begin{aligned} \omega_1 &: \ x = 1 \\ \omega_2 &: \ x = 2 \\ \omega_3 &: \ x = 3 \end{aligned}$$

Clearly, x cannot be 1, 2 and 3 at the same time. If any one of the constraints is removed, the two remaining constraints will still be infeasible. However, if two constraints are removed, the remaining constraint will be feasible.

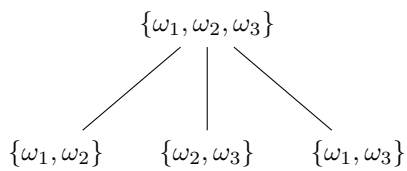

Figure 3.6: Lattice of the *Unsat Core* set of Scenario 2.

In Figure 3.6 is shown the lattice of of the *Unsat Core* set, where there are three lower bounds, hence three minimal *Unsat Cores*.

Scenario 3

In the third scenario four constraints are grouped into two pairs of constraints in conflict with each other:

$$\begin{aligned}\omega_1 &: \ x = 1\\ \omega_2 &: \ x = 2\\ \omega_3 &: \ y = 1\\ \omega_4 &: \ y = 2\end{aligned}$$

As can be seen, constraints ω_1 and ω_2 can never be true at the same time. The same holds for constraints ω_3 and ω_4. Even if one of the constraints is removed, there is still a remaining pair of infeasible constraints. Only if at least one constraint of each of the pairs is removed, does the problem become feasible.

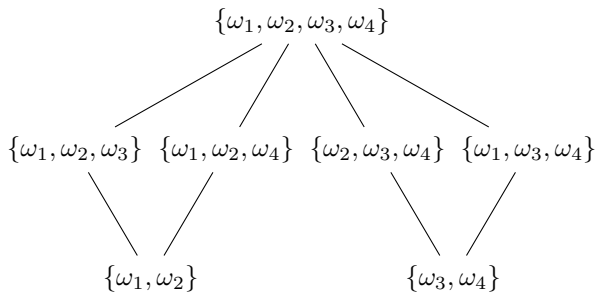

Figure 3.7: Lattice of the *Unsat Core* set of Scenario 3.

Figure 3.7 shows that there are two minimal *Unsat Cores*, each with cardinality 2, namely ω_1 and ω_2, and ω_3 and ω_4.

The three scenarios presented in this section show that even small formulas can easily have multiple *Unsat Cores* and such *Unsat Cores* can intertwine in different ways. Being able to catch them all could provide important information about the system modelled by the infeasible formula and therefore have interesting applications, as shown in Paper E.

3.3 Modelling using MILP/SMT

In the previous section we give a brief introduction to the techniques used by MILP and SMT solvers. We discussed how SMT solvers allow for logical conditions over literals belonging to different theories. We now show how these approaches can be used to deal with problems in industrial systems. As an example we model the non-overlapping of operations sharing the same resource, first as MILP and then as SMT.

One way to formulate the constraints is by having a continuous variable representing the start time of each operation. Some operations are supposed to be executed by the same machine and therefore cannot overlap in time; the first operation has has to be completed before the next one can start.

Let s_a and s_b be continuous variables representing the starting time of operations a and b, and let d_a and d_b be the duration of operations a and b, respectively. For a MILP solver to handle the *non-overlap* constraint, it is necessary to declare an additional binary variable $z \in \{0, 1\}$ and find a large enough number M. If M is not large enough the solver may return the wrong schedule. The constraints to declare are then:

$$s_a \geq s_b + d_b - M \cdot z, \tag{3.11}$$

$$s_b \geq s_a + d_a - M \cdot (1 - z) \tag{3.12}$$

When $z = 1$ (and M large enough), constraint (3.11) is trivially fulfilled, since $s_a \geq 0$ and $s_b + d_b - M < 0$, and thus this constraint has no effect. At the same time, when $z = 1$ (3.12) becomes $s_b \geq s_a + d_a$, which puts a requirement on s_b to be larger than the staring time of operation a plus its execution time (that is, operation a's end time). On the other hand, when $z = 0$, (3.12) is trivially fulfilled while (3.11) becomes $s_a \geq s_b + d_b$, thus constraining s_a to be larger than s_b plus its execution time.

On the other hand, with an SMT solver it is possible to combine literals of linear arithmetic, i.e. linear inequalities, with propositional logic:

$$s_a \geq s_b + d_b \vee s_b \geq s_a + d_a.$$

There can be much harder logical conditions to model and it becomes harder and harder to do it with a MILP solver since everything has to be in the form of a linear inequalities. Also, the additional binary variables used to model logics

increase the search-space size exponentially; in a model counting n binary variables, a branch and bound algorithm has to solve 2^n relaxed problems in the worst case, only to account for the binary variables, though there may be other integer decision variables that increase the size even further.

3.4 Comparison of MILP and SMT over benchmark problems

The benchmark problems presented in Chapter 2 have been modelled as MILP and SMT and solved using the state-of-the art MILP and SMT solvers, Gurobi [79] and Z3 [73], respectively. In this section the solutions provided by the MILP and SMT solvers are compared in terms of solving time.

The Job Shop Problem

As for all classes of problems, the model formulation plays an important role in the solving phase so it is important to know which are the main formulations existing to model the JSP and how they affect the solver's performance. In [80], the authors present the *disjunctive*, *time-indexed*, and *rank-based* models as the most common formulations for the JSP (an in-depth description of these models is provided in Paper A) and present a computational analysis over a set of benchmark instances. Such formulations are tested using the state-of-the-art MILP solvers CPLEX [81] and Gurobi and, for both solvers, the *disjunctive* model shows the best performance.

In Paper A, the above mentioned models are formulated as SMT and then they are evaluated on a set of benchmark problems using Z3. The comparison shows that also for SMT solvers the *disjunctive* model outperforms the other two formulations, sometimes by an order of magnitude. This allows the disjunctive model to solve much larger problem instances in reasonable time, as shown in the tables of Paper A. For a fair comparison, the *disjunctive* model is formulated as MILP as well and run over the same set of problems, using Gurobi and the SMT solver OptiMathSAT [82]. These tests show that Z3 outperforms Gurobi on all problems and the gap increases as the problem size increases. In general, the combination of an SMT solver and the *disjunctive* model is able to handle relatively large benchmark instances in a relatively short time (timeout set to 20 minutes). In Figure 3.8, two cactus plots depict

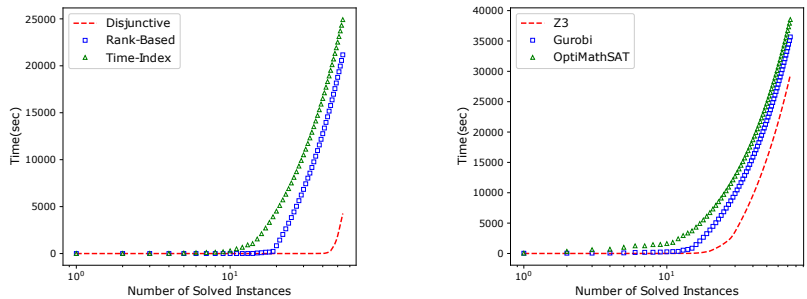

Figure 3.8: Comparison of different model formulations (left) and different solvers (right) over a benchmark set of the JSP.

the comparison of models (left) and the comparison of solvers (right) in terms of number of solved instances versus running time. Further investigations [61] revealed that, when expressing the JSP using the disjunctive model, the problem's constraints fall into the domain of *difference logic*, a fragment of linear algebra; Z3 uses the polynomial Ford-Bellman algorithm [83] to check whether an assignment for this theory is feasible or not. Having the SAT engine efficiently finding feasible assignments and the theorem prover quickly checking them leads to an overall fast execution.

The Bin Sorting Problem

The original problem formulation for bin packing and bin covering [26] involves binary variables to model whether an item is placed into a bin or not. When formulated in this way, the symmetry of the problem leads to a very large search space for relatively small instances that makes it hard to quickly compute optimal solutions.

An alternative formulation involves the enumeration of the possible combinations of items that fit into a bin. In this *enumerative formulation* each variable is an integer representing one combination and the value it is assigned is the number of bins filled with the items included in the combination. This formulation can break the problem symmetry but nonetheless the number of possible combinations grows exponentially with the number of items and the problem soon becomes intractable. However, in Paper B we show that only a

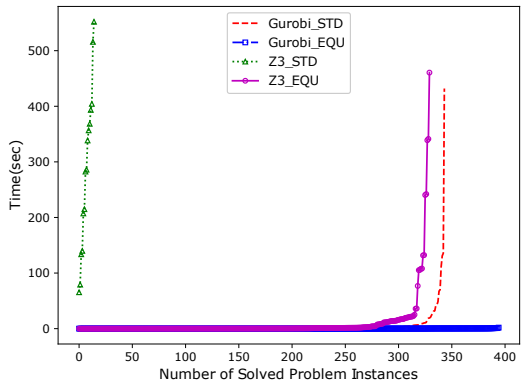

Figure 3.9: Comparison of different model formulations using Z3 and Gurobi over a benchmark set of the BCP.

small portion of all these combinations can actually form an optimal solution, and this allows to use the enumerative formulation to solve problems more efficiently.

Paper B is an extension of our previous work on the subject presented in Paper F, where we implemented such an enumerative approach for the BCP only, but for MILP and SMT (in Paper B only MILP is used). For this problem the MILP solver Gurobi proved to be much faster than the SMT solver Z3 which is the reason why SMT is not used in Paper B. The problem formulation does not involve complex logical constraints, rather a conjunction of linear inequalities and therefore using an SMT solver did not provide any advantage in the modelling phase, nor in the solving one. In Figure 3.9 the cactus plot compares Z3 and Gurobi running the standard formulation from [26] (STD) and the formulation presented in papers B and F (EQU).

The Vehicle Routing Problem

As a first step toward the formulation of the CF-EVRP, we investigated the *Vehicle Routing Problem with Time Windows* (VRPTW). This problem is a VRP where each customer must be served within a time window. Also, the VRPTW belongs to the class of *capacitated* VRPs, where customers have requirements on the amount of goods to be delivered, namely *demands*, and

vehicles have limits on the amount of goods they can deliver, namely *capacity*.

Traditionally, the problem is modelled using a set of *three-index* binary variables to keep track of which vehicle is travelling from A customer to customer B. Moreover, additional variables are required to keep track of the load carried by the vehicles and the service time of the customers. Such formulation is called *three-index formulation*, since variables are identified by three indexes, one denoting the customer the direct travel starts from, one denoting the customer it ends at, and one denoting the vehicles travelling. However, a more recent formulation by [25] only requires a set of *two-index* binary variables indicating a direct travel from customer A to customer B, i.e., no vehicle is specified. In fact, because exactly one vehicle is allowed to visit each customer, there is no risk of ambiguous solutions even when the third index, that specifies the vehicle, is omitted. This formulation helps to break the symmetry of the problem.

We implemented the two-index formulation both as MILP and SMT and compared their performance over the benchmark set of Solomon instances [58] counting 25 and 50 customers. The goal is to minimize the total travelled distance. The experiments are performed on an *Intel Core i7 6700K, 4.0 GHZ, 32GB RAM* running *Ubuntu-18.04 LTS*; the solvers compared are Z3 4.8.9 and Gurobi 9.0.0. The Solomon instances are grouped into three classes, based on the customer distribution, i.e. *clustered* (c), *random* (r), or mixed random and clustered (rc); they are also further divided into *short time horizon* (100) and *long time horizon* (200) (see more on time horizon in the next chapter).

Table 3.1: Comparison of Z3 and Gurobi over Solomon benchmark instances with 25 and 50 customers. The time limit is set to 1200 second; for each class of instances it is reported the average time to solve the instances that did not timeout, and the number of instances solved out of the total number of instances in that class. The symbol "-" means that no instance is solved for a specific class.

	Gurobi				Z3			
Customers	25		50		25		50	
Instance	Av. (s)	Opt	Av. (s)	Opt	Av. (s)	Opt	Av. (s)	Opt
c100	80.5	9/9	200.9	9/9	42.1	5/9	836.6	2/9
c200	75.9	8/8	102.6	8/8	45.5	3/8	754.8	1/8
r100	157.9	12/12	481.2	12/12	25.6	1/12	-	0/12
r200	61.7	11/11	525.4	11/11	129.8	1/11	-	0/11
rc100	301.8	8/8	527.2	8/8	391.6	2/8	-	0/8
rc200	312.2	8/8	525.4	8/8	36.9	1/8	-	0/8

Table 3.1 shows the average running time to solve the instances of each class to optimality within the time limit of 1200 seconds, and the number of solved instances (out of the total). Gurobi is able to solve all instances within the time limit both for 25 and 50 customers, while Z3 could only solve a few. Moreover, although in most cases the average solving time for Gurobi is larger than the solving time for Z3, it is important to remember that it is calculated only over the solved instances; hence some hard-to-solve instances took Gurobi long to solve and were not solved at all by Z3, but the timeout does not show up in the average. In conclusion, Gurobi shows better performance than Z3 in dealing with VRPTW.

3.5 Conclusions on the MILP/SMT performance

In the previous section we have introduced the performance evaluation of the state-of-the-art MILP solver Gurobi and the state-of-the-art SMT solver Z3 over benchmark problems of three optimization problems. Although these two solvers do not reflect the general performance of all the solvers from their respective community, it is possible to draw some conclusions about their expected behaviour and how it relates to certain problem structures.

For instance, the BSP and the VRPTW models involve only conjunctions of linear inequalities. Therefore the MILP solver has no trouble dealing with very large models (in the evaluation presented in Paper B, we generated models counting up to twenty million variables). The SMT solver also comes with theories to handle linear algebra, but it is not as efficient as the MILP solver and, therefore, slower. One reason for this is that companies such as Gurobi and CPLEX have been spending decades refining the algorithms that run under the hood of their MILP solvers, while SMT is a relatively newer approach.

On the other hand, the JSP has a disjunctive constraint (non-overlap of operations requiring the same job) that has to be modeled using additional binary variables together with the *big M* method when using the MILP solver. The SMT solver, instead, will set up a SAT problem where each linear inequality is treated as a Boolean variable and find an assignment (if such exists) for them; then the theory prover will check the feasibility of the assignment of the Boolean variables for the corresponding inequalities in polynomial time using the Bellman-Ford [83] algorithm.

The computational analysis run for the JSP, the BSP, and the VRPTW seems to indicate that, if the problem structure involves only conjunctions of literals belonging to linear algebra, the MILP solver will likely be the most suitable option; on the other hand, if the problem involves more logical constraints, such as disjunctions, implications, etc., the SMT solver will likely outperform the MILP solver, being naturally designed to handle constraints of propositional logic over literals of different theories. These conclusions are used in Chapter 5 to reason about which solver is the best candidate to solve each sub-problem resulting from the decomposition of the CF-EVRP.

CHAPTER 4

The Conflict-Free Electric Vehicle Routing Problem

In this chapter the CF-EVRP is formally defined and a discussion on the different requirements is presented. Essentially, the CF-EVRP is the problem of providing a high level schedule for a fleet of AMRs (often simply called *vehicles* in the following discussions) to pick up and deliver (tasks) goods through an industrial plant. Such a schedule involves the assignment of vehicles to the tasks and the computation of the paths to travel to execute the tasks. The problem does not include low-level trajectory planning for the AMRs, which is handled by a separate system. The main reason for this is that the information necessary to solve the low-level trajectory planning problem, including current and estimated future states of dynamic obstacles, is not available at the time of solving the scheduling problem. Instead, the approach defines the *capacity* of the road segments to differentiate areas where the AMRs can maneuver around each other from areas where they can not.

When solving the CF-EVRP, the plant and its workstations, storage, and depots are abstracted into a strongly-connected, weighted, directed graph. A solution to the CF-EVRP is therefore a list of nodes and edges for the selected AMRs; for each node and edge it is also reported when the AMR will arrive to it (node), or start travelling through it (edge).

4.1 Formal definition of the CF-EVRP

The CF-EVRP is an extension of the VRP. The motivation for the formulation of the CF-EVRP is the need for managing a fleet of AMRs whose task is to pick up and deliver material in a manufacturing or assembly plant. Unlike other scenarios, where it can be assumed that roads between customers are wide enough to allow vehicles to travel freely through them, without having to worry about other vehicles travelling through the same roads, a plant layout may have areas where only a limited number of vehicles can be accommodated simultaneously. In other words, the geometry of certain roads may be such that two or more vehicles may block each other's way, hence generating a conflict. For this reason we define the *capacity* of the road segments, intersections, and workstations and include *capacity constraints* in the problem formulation. A schedule is said to be *conflict-free* if it fulfills the capacity constraints.

Moreover, transportation vehicles in industrial plants may be powered by batteries, and so have a limited operating range. It is therefore important to consider that vehicles may have to return to the depot to charge their battery after having travelled a certain distance. This means that the vehicle will not be available for some time, depending on charging related parameters.

Finally, customers may be related by precedence constraints. A vehicle must serve them in a pre-defined order, since one or more customers may represent storage locations to pick material from and another customer may represent the workstation to deliver material to. These and further requirements of the CF-EVRP are listed into the following:

- Customers in the CF-EVRP represent *jobs*. Jobs are sets of *tasks*, typically one or more pick up tasks and one delivery task;
- All jobs have to be completed for a successful schedule; for a job to be completed a vehicle has to be assigned to it and visit the locations of the job's tasks according to the tasks' sequence and within their respective time windows.
- Vehicles are not allowed to arrive at the task's location before the time window's lower bound and wait there (many other VRP formulations, allow such waiting).
- Vehicles are powered by batteries with limited capacity but with the ability to recharge at the depots. It is assumed that state of charge

increases proportionally to the time spent at the depot and decreases proportionally to the travelled distance. Also, vehicles travel at constant speed v or they are stationary.

- The round-trip between the depot and a customer is shorter than the vehicles operating range, therefore vehicles can always serve at least one customer and go back to the depot before running out of charge.
- There exist multiple depots; vehicles have to return to the depot they were dispatched from and can only recharge their batteries there (without queuing).
- A non-empty subset of vehicles is eligible for each job.
- All vehicles have the same operating range and start at full charge; whenever they return to the depot they charge to full state-of-charge before becoming available again.
- Road segment capacities limit the number of vehicles a road segment can simultaneously accommodate at any point in time.
- Only (non-cyclic) *paths*, that is, only finite sequences of edges that join sequences of distinct vertices are considered.
- The *total travelled distance* is used as optimization criterion to evaluate the quality of the solutions.

4.2 Customers Serving

When it comes to serving the customers, the CF-EVRP shares common traits with other VRPs. First, the existence of a time windows, which is the main feature of the VRP with Time Windows (VRPTW). The VRPTW is an extension of the VRP where customers have to be served within given time window, a time interval that spans from the *earliest* to the *latest* arrival time. Vehicles are allowed to arrive at the customer's location before the earliest arrival time, but they will have to wait to serve the customer [84]; on the other hand, they are not allowed to arrive later than the latest arrival time. The CF-EVRP is also characterized by precedence constraints among customers; this feature relates it to the VRP with Backhauls [85], where the set of customers is divided

into a subset of *linehaul* customers, each requiring a given quantity of product to be delivered, and a subset of *back haul* customers, where a given quantity of inbound product must be picked up. Also, in CF-EVRP there can be multiple depots, which is used in many variants of VRPs and was first introduced in [86]. There exists problem formulations where vehicles are allowed to start and end their routes at different depots [87]. however, in the CF-EVRP, we follow the standard formulation of multi-depot VRPs [88], where vehicles are restricted to end at the depot they started from. Finally, not all vehicles are eligible to serve a particular customer; instead, for each customer, there exist a subset of the full fleet that includes the eligible vehicles to serve such customer. This feature is studied in the Heterogeneous VRP (HVRP) [89] and is motivated by the need of specific vehicles to execute certain tasks.

4.3 Limited Operating Range of the Vehicles

The first attempts to model rechargeable vehicles date back to the late 90s, [90], and models were subsequently extended in [91] and [56]. When representing the problem as a graph, charging stations are special nodes where the vehicle can charge its battery on its way to customers. Since in a VRP vehicles have to visit customers exactly once, a solver for a VRP problem may misjudge an instance, i.e., declare it unfeasible while it is actually feasible, because a feasible solution could require the same vehicle to visit the same charging station twice or more, or not at all. In order to avoid these situations, it is possible to define *virtual* charging stations, i.e., copies of the actual ones located in the same spot, so that the same vehicle can go back to the same location to recharge without having to visit the same node more than once. For the virtual stations the requirement of visiting them exactly once is then relaxed to *at most once* so that vehicles do not have to visit them unless they need to. In [56] a model formulation based on the concept of virtual charging stations to implement a hybrid local and tabu search is presented. The model has been extended further by [92], to include additional requirements on the cost function to reduce the energy consumption. Ever since, due to the faster and faster spread of electric vehicles as well as car sharing services, there has been a large effort in finding scalable algorithms for the Electric-VRP (E-VRP). In [93], a hybrid Adaptive Large Neighbourhood Search is proposed. In [94] the focus is on optimizing the vehicle distributions and fleet sizes for

Mobility-on-Demand systems. In [95] models and coordination policies for fleets of self-driving vehicles combined with public transit are discussed.

In the CF-EVRP, the recharge of vehicles' batteries is only allowed at the depots the vehicles were originally dispatched from and the vehicles have to fully charge their batteries before being available to serve more customers. This restriction helped to design an efficient algorithm to schedule the fleet of AMRs (more details on this topic in Chapter 5). It is also reasonable in an industrial context that vehicles have enough charge to serve one or more customers before needing to recharge.

4.4 Travelling Between Customers: the Paths Choice

Let us assume that a VRP represents the delivery of goods performed by a truck company in a country, where the customers may be different cities. When travelling between two cities, the driver may take little detours, due to road closure, or lunch breaks. However, as long as these detours are negligible compared to the whole trip, the total distance is not affected so much as to affect the schedule. Therefore the path between two customers can be abstracted into a straight line and minor detours will not affect it. When paths between customers can be abstracted into straight lines, the input to a VRP is usually represented by a *complete graph* [84], [88], [96], i.e., an undirected graph in which every pair of distinct vertices is connected by a unique weighted edge. This means that vehicles are allowed to travel directly from any customer to any other customer at the cost of the edge that connects the two nodes representing the customers. It also implicitly implies that the actual location of the vehicle is not known.

This approximation works well for many applications. However, in a plant layout, omitting the path used to travel between two locations would cause loss of important information. In fact, it is necessary to know exactly what road segments the vehicles are travelling on in order to avoid conflicts and because detours to avoid congestion could cause significant delays on the delivery and cause to miss the time window. For this reason the whole plant layout and its workstations, storage areas, and depots are abstracted into a strongly-connected, weighted, directed graph. In such a graph the nodes represent the workstations, the intersections of drivable road segments, and the depots;

nodes can usually accommodate only one vehicle at the time, unless they are *hubs*, in which case they can accommodate an arbitrary number of vehicles at the same time. The edges represent the road segments; each edge has two weights representing respectively the segment's length and its capacity. The capacity of a road segment can have two values; *1* if the edge is too narrow for two vehicles travel through it in opposite direction at the same time, *2* otherwise. An example of a plant abstracted into a graph is given in Figure 4.1 taken from Paper D, where the road segments length is the edges weight (normal font) and the segments' capacity is the weight in subscript font; for instance, edge $(1, 2)$ has a weight of $17,5_1$, meaning that its length is 17.5, and its capacity is 1.

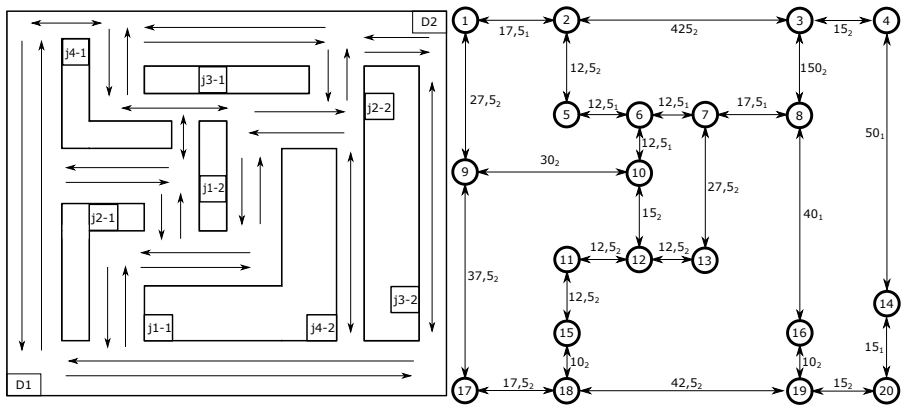

Figure 4.1: A hypothetical plant layout taken from paper D and its abstraction into a graph.

Designing conflict-free routes in environments with limited capacity of road segments is a problem that has been investigated over the years. One of the first attempts to tackle the problem is presented in [97], where the solution is computed by means of column generation. In [98], an ant colony algorithm is applied to the problem of job shop scheduling and conflict free routing of mobile robots. In [99], a collision-free path planning for multi AGV systems based on the A^* [100] algorithm is presented. In this work, the environment is modeled as a grid, and conflicts can originate from vehicles occupying the same spot on the grid at the same time. In [101], a heuristic approach to solve the conflict-free routing problem with storage allocation is presented. In [102], a MILP formulation to design conflict-free routes for capacitated

vehicles is presented. This is an exact method, but it can only solve relatively small problem instances. In [103] a hybrid evolutionary algorithm to deal with conflict-free AGV scheduling in automated container terminals is presented.

4.5 The Intrinsic Complexity of the CF-EVRP

As we have seen in the previous sections, the CF-EVRP shares common traits with many other VRPs and its features have been tackled by others in past and recent times. However, it is the combination of these features in one *large* problem that further complicates the solution process and requires the design of a new procedure to compute a schedule for the AMRs fleet in reasonable time. On the other hand, the CF-EVRP offers a general framework to model a large class of VRPs, since it allows for heterogeneous fleets, limited operating range, time windows, precedence constraints among customers and capacity constraints on the road segments. Finding an efficient procedure to solve it may contribute significantly to the field of VRPs. Moreover, to the best of our knowledge, there is no work focusing on all these features at once. With this work, we have the opportunity to fill this gap in the literature and at the same time solve a relevant industrial problem.

In the next chapter is presented the proposed solution method for the CF-EVRP in Chapter 5.

CHAPTER 5

A Compositional Algorithm to solve the CF-EVRP

In this chapter we present the proposed method to solve the CF-EVRP. First, we introduce the mathematical formulation of the problem, presented in Paper C, which is formulated as SMT and solved using the optimizing SMT solver Z3. Then we present an alternative formulation, where the CF-EVRP is decomposed into sub-problems that can be used by the compositional algorithm ComSat to find a solution to the original problem. Finally, the inefficiencies of ComSat are analyzed and an improved version of it that makes use of the *Unsat Core* is presented.

5.1 A Mathematical Formulation of the CF-EVRP

Paper C presents a complete, monolithic formulation of the CF-EVRP. In order to compute a conflict-free solution for the CF-EVRP it is necessary to know where each vehicle is at any time; therefore, the monolithic formulation (henceforth called MonoMod) involves time and space discretization. It is well known within the community of scheduling and optimization that discretization often leads to a large increase in the search space size for a relatively small increase in the problem instances' size. Thus, discretization is not suit-

able for large systems involving a high number of vehicles or customers, or a vast plant. Still, to test the performance of the state-of-the-art SMT solver Z3 for CF-EVRP, MonoMod was formulated as SMT and solved by Z3.

Given that the CF-EVRP is novel, there are no problem instances available for benchmarking. Hence, we generated sets of instances to evaluate the performance of MonoMod by measuring the solving time with respect to the parameters of the solved instances. As mentioned before, the plant is abstracted into a strongly-connected, directed, weighted graph. The graphs in the generated benchmark set can have 15, 25, or 35 nodes, which correspond to a grid-like graph of size 5×3, 5×5, and 5×7, respectively. The connectivity of the graph also affects the instance size, hence we defined the parameter *Edge Reduction* to represent the number of edges. The value *0* correspond to a complete grid; the value *25* corresponds to a grid where 3, 6, or 9 pairs of edges (i.e. (A, B) and (B, A)) are removed for N equal to 15, 25, and 35, respectively; the value *50* corresponds to a grid where 6, 12, or 18 pairs of edges are removed for N equal to 15, 25, and 35, respectively. Figure 5.1 shows the 5×3 graph with edge reduction coefficient equal to 0 (a), 25 (b), and 50 (c). For the 5×5 and the 5×7 graphs the same pattern is repeated once and twice, respectively. The number of vehicles available and jobs to execute also affects the instance's size. Finally, because of time discretization, also the *time horizon* contributes to the instance size. Typically the time horizon is defined a as fixed point in time in the future when all jobs are required to be executed; however, in CF-EVRP it represent the time when all jobs are required to be executed and the vehicles are required to have returned to the depot they were dispatched from. This feature is captured by constraint (C.10), though not explicitly stated in Paper C.

In the generated instances the capacity of all road-segments is restricted to *1*. For the graph it is specified which nodes are *hubs* that can accommodate an arbitrary number of vehicles at the same time. Another parameter of the instance is the number of vehicles available. Each vehicle is assigned a starting node and more vehicles can start from the same node (in this case the node must be a *hub*). Also, a list of jobs to execute is given; each job is characterized by a number of tasks and the vehicle(s) eligible to execute it. In turn, each task is characterized by a location (node), a precedence list specifying the tasks to execute before (if any), and a time window. Other parameters of the instance are the charging and discharging coefficients, as well as the operating

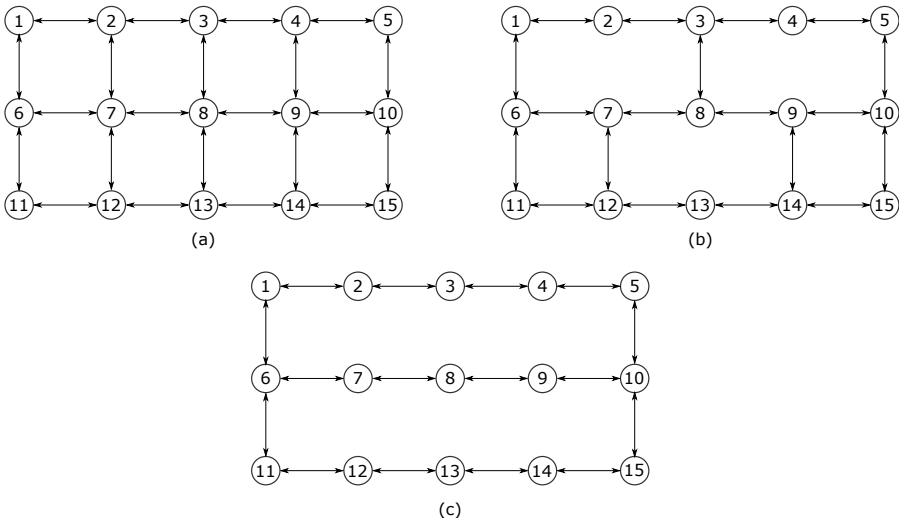

Figure 5.1: Graph with 15 nodes arranged in a 5×3 grid with edge reduction coefficient equal to 0 (a), 25 (b), and 50 (c).

range of the vehicles, and the time horizon by when all jobs should be finished and the vehicles should be back at their starting locations.

In the computational analysis 160 problem instances are evaluated. The parameters'values used for the problem generation are:

- *N-V-J* (nodes, vehicles, and jobs, respectively). *N-V-J* can be either *15-3-5*, or *25-4-7*, or *35-6-8*. Each job is composed by one pickup and one delivery;
- *Edge Reduction* can be either *0*, or *25* or *50*.
- Time Horizon can be either *15*, *20*, *25*, or *30*.
- All edges have weight equal to *1*.

For each combination of the above-mentioned parameters, five different problems are generated by randomly assigning the starting node for each type of vehicle, the operating range, the charging coefficient, the number of vehicles available per each type, the type of vehicle required for each job, and the location of the jobs' tasks. For the analysis we used Z3 4.8.9. The time limit

is set to 10800 seconds (three hours). All the experiments were performed on an *Intel Core i7 6700K, 4.0 GHZ, 32GB RAM* running *Ubuntu-18.04 LTS*. Though Z3 allows for optimization of the objective function, the size of the problems evaluated is such that no optimum is expected to be found in any reasonable time. Therefore Z3 is set to find satisfiable (hence sub-optimal) solutions.

Table 1 of Paper C shows the results of the computational analysis. As expected, for larger values of time horizon, nodes, vehicles, and jobs the computation time increases, soon becoming unreasonable for a real-time scheduler. Already for problems counting four vehicles and seven jobs it can take up to two hours to compute a feasible schedule. Moreover, the models are so large that only generating them can take several minutes. It is clear that the monolithic approach cannot be used for large scale problems, and a more efficient solution is required.

5.2 A Compositional Algorithm to Solve the CF-EVRP

In this section we present ComSat, a compositional algorithm to solve the CF-EVRP. As mentioned many times before, we decompose the overall problem into sub-problems. Now we provide explicit descriptions of, and thorough discussions on these sub-problems. In Paper D, a mathematical formulation for each of these sub-problems is also presented. Then, we show how ComSat can find a solution to the CF-EVRP by iteratively solving these sub-problems.

As mentioned in chapters 1 and 2, when a problem is too time-consuming to be solved by a monolithic procedure, a possible solution is to decompose it into smaller problems and then build an overall solution using the solutions to these sub-problems. For the CF-EVRP, a major challenge is caused by the need of producing conflict-free solutions. Since time discretization is needed to model the location of vehicles at each time, the number of variables and constraints to declare grows exponentially with the instance size. Instead, the strategy adopted with ComSat is to group the different requirements stated in Chapter 4 and solve them sequentially, backtrack when an intermediate solution is infeasible and look for a different one. In this way it is possible to avoid time discretization and solve larger instances in shorter time.

The Main Sub-Problems

A pre-processing step is required before the actual sub-problems can be solved, the *computation of the shortest paths* between any customer and the depots and between any two customers. In fact, in order to solve the first sub-problems, it is necessary to have a unique path to travel from any customer (or depot) to another.

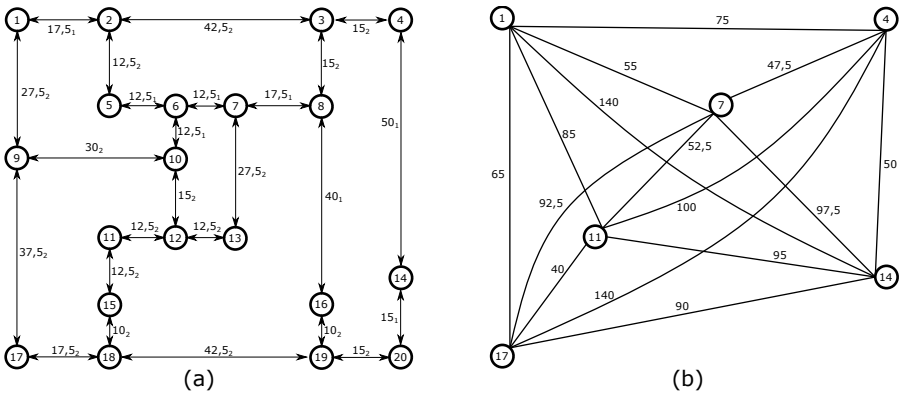

Figure 5.2: Computation of shortest paths between any two customers/depots.

Let us clarify this point. Figure 5.2 shows a graph abstracting a hypothetical plant (a). Let us assume that nodes *17* and *4* are depots and nodes *1*, *7*, *11*, and *14* are customers. In order to turn the problem into a VRP it is necessary to know the distances among each pair of nodes, and such distance must be unique. In (b) it is shown how each of the above mentioned nodes is now connected with any other node by an edge whose length is the length of the shortest path to travel between those pairs of nodes in (a).

The first sub-problem is the *Routing Problem*. This is a multiple depot VRP with time windows, a heterogeneous fleet of vehicles and precedence constraints among the customers. Additionally, the routes are restricted to be at most as long as the operating range of the vehicles; by adding this restriction, we eliminate the complexity of dealing with the limited operating range of the vehicles, which is then tackled in the next sub-problem. Whenever solving the *Routing Problem*, only the shortest paths to connect customers/depots with each other are considered; in fact, at this stage, no capacity constraint is included. Also, at the moment, no actual vehicle is considered; routes are

designed to be assigned to existing vehicles, which means that customers are grouped based on the types of vehicles eligible to execute them, but no specific vehicle is assigned to them. This is an optimization problem and the parameter to minimize is the number of routes to serve all customers. We chose this criterion because we assume that the fewer vehicles are dispatched the fewer the chances of running into conflicts. The solution to this problem is a set of routes to serve all customers according to the requirements mentioned in this paragraph. Moreover, for each route, the *latest start time* is computed. This is the latest time for the vehicle assigned to the route to set off such that the strictest time window of the route is met. Figure 5.3 (b) shows a possible routing solution for the problem presented in Figure 5.2 (a).

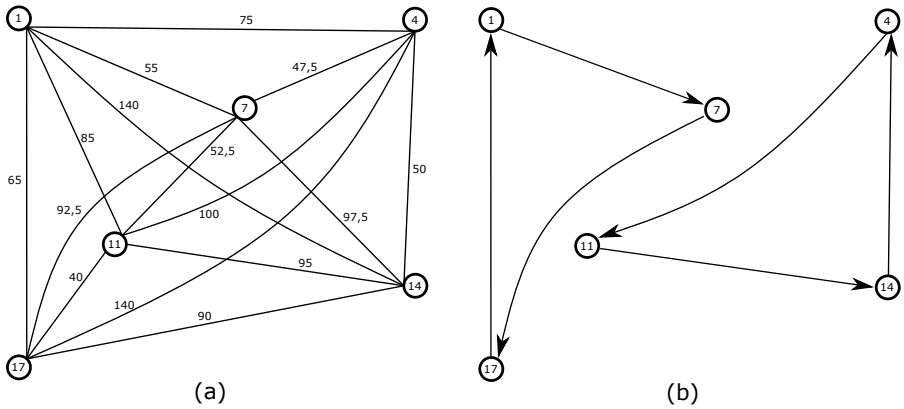

Figure 5.3: Feasible solution to the *Routing Problem*.

The second sub-problem is the *Assignment Problem*. This is the problem of deciding which vehicle should be assigned to each of the routes designed by solving the previous sub-problem. Vehicles are assigned based on their type, but also on their availability; in fact, if a vehicle was dispatched to serve some customers, once it returns to its depot it will not be available until its battery is fully charged. The time required for the charging, based on our assumption, is linearly proportional to the length of the route travelled before charging. Assignments are also decided based on the time windows of the customers that are served in a certain route. Of all the eligible vehicles, the choice is made among the ones that are available early enough so that the time windows can be met. This is a feasibility problem, whose solution is the assignment of one

vehicle to each route. Moreover, for each assignment is specified the *latest start time* for the vehicle to execute the route such that the strictest time window in the route is met. The problem is modelled as a JSP, where vehicles are *resources* and routes are *jobs* (see Chapter 2 for reference to the JSP).

The third sub-problem is the *Capacity Verification Problem*. This is the problem of verifying that the vehicles do not break the capacity constraints while travelling through the plant to serve the customers belonging to the routes they are assigned to. In order to do so, it is necessary to know exactly what nodes and edges the vehicles are passing by. This information is available from the pre-processing step, when the shortest paths between any two customers/depots were computed. Each route is therefore represented by a sequence of nodes (and edges between the nodes) that starts at the node representing a depot, passes by the nodes representing the customers belonging to the routes, and returns to the depot. Figure 5.4 shows how the routes computed in the previous phase (a) are then "plugged" into the actual plant layout and checked for capacity constraints violation (b).

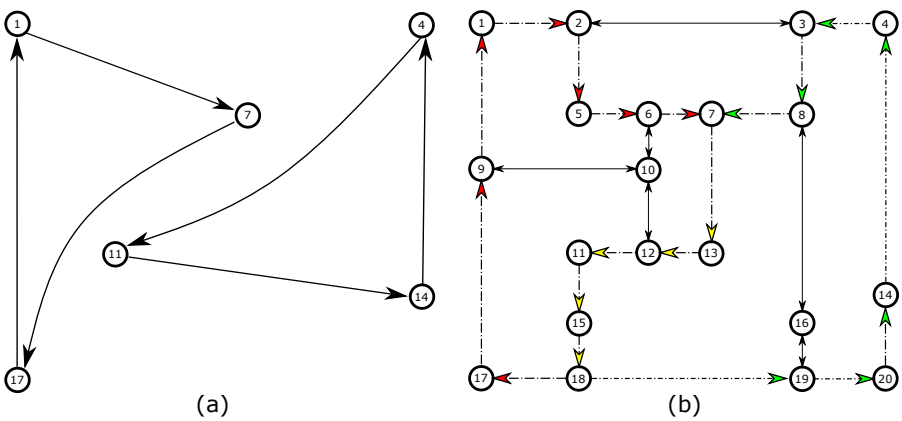

Figure 5.4: Routes (a) are verified against capacity constraints on the actual plant layout (b).

One route is marked with red arrows while the other is marked with green ones. The yellow arrows represent the edges that are used by both routes. Based on the tasks' time windows, it is possible to infer whether the two routes can use the edges (and the nodes in between them) at different times (conflict-free) or must use at least one of them at the same time, thus generating a conflict. This is a feasibility problem, whose solution is, for each route, the

arrival time at each node included in the route. This problem is also modelled as a JSP, where nodes and edges in the graph are *resources* and the routes are the *jobs*. The vehicles assigned to the routes have to use nodes and edges to move through the plant and serve the customers. In order to enforce the capacity constraints, the nodes and edges with limited capacity are treated as resources that can only execute one job at the time. Moreover, the deadline for the customers of each route apply to the node where the customer is located.

Search of Alternative Paths

In the previous section the main sub-problems that the CF-EVRP is decomposed into are presented. They are solved sequentially and, if all of them are feasible, the solution to the *Capacity Verification Problem* is a solution to the CF-EVRP. However, if the *Capacity Verification Problem* is not feasible, it is necessary to change the paths by solving the *Paths Changing Problem*. This is the problem of finding the shortest paths that have not been selected before to connect any two customers/depots. It is an optimization problem and the reason for finding the shortest paths is that the shorter the paths, the faster the vehicles can travel from customer to customer, the higher the chances they will meet the time windows. Figure 5.5 shows how the path that connects Node *7* to Node *17* is initially $7-13-12-11-15-18-17$ in (a) and is replaced by $7-6-10-9-17$ having the same length in (b).

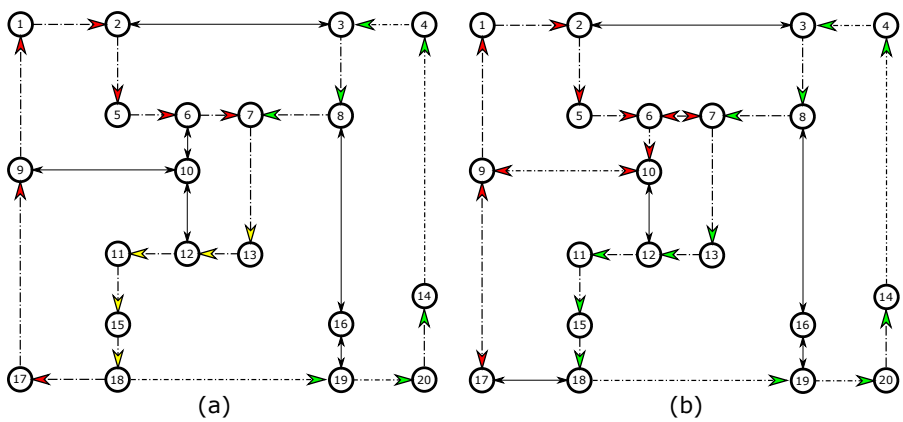

Figure 5.5: The shortest paths used to design routes (a) are changed to avoid capacity constraints violations (b).

Now the red and green routes only overlap on Node 7, thus minimizing the chances of a conflict.

However, after a new set of paths has been computed, there is no guarantee that the *Routing Problem* will still be feasible. It could be the case that the previously computed routes now exceed the vehicles operating range, or that the vehicles can no longer meet the time windows. It is therefore necessary to check the feasibility of the previously computed routes using the new set of paths by solving the *Routes Verification Problem*.

Mathematical Formulation of the Sub-Problems

When modelling a problem, there is often more than one way to formulate it, based on what variables are chosen to describe its dynamics. The formulations of the sub-problems presented in Paper D are meant to exploit the strengths of the solvers used to solve them. In particular, when formulating the *Assignment* and *Capacity Verification* problems, the choice of modelling them as JSP is motivated by the findings of Paper A. Also, the formulation of the *Path Changing Problem*, inspired by [104], suits SMT solvers as it only uses Boolean decision variables and propositional logic to model the constraints. As for the *Routing Problem* the choice of modelling it as MILP is motivated by the results of Chapter 2. Finally, the *Routes Verification Problem* is also formulated as SMT; even for large instances, this sub-problem only involves a small number of variables and constraints and can be quickly solved by both SMT and MILP solvers. Therefore there was no thorough investigation on whether one solver would be faster than the other.

ComSat

After introducing the sub-problems, let us discuss how a solution to the CF-EVRP is found by solving them, or if no solution exists how it is possible to conclude so. ComSat being a *compositional algorithm* means that it is composed by sub-algorithms, each solving one of the sub-problems listed in the previous sections. Table 5.2 shows which sub-algorithm solves which problem, what input(s) they take, and what output they return. The glossary of Table 5.1 summarizes the names of the sets used to store and exchange information among the sub-problems.

The algorithm, whose flowchart is shown in Figure 5.6, begins with the

Table 5.1: Glossary for the sets of the sub-problems.

CP: set of current paths
SP: set of shortest paths
NP: set of new paths
PP: set of previous paths
CR: set of current routes
PR: set of previous routes
CA: set of current assignment of vehicles to routes
PA: set of previous assignment of vehicles to routes
CVS: set of *conflict-free* routes (pairs of nodes and arrival times for each route)
RVF: Boolean variable representing the feasibility of the *Routing Problem*

computation of the shortest paths *SP* between each pair of tasks. This step is only executed once to provide unique paths for the *Routing Problem*, which is then solved. In this step neither the vehicles' availability nor the segment capacities are considered; the goal is simply to design routes to serve tasks within the time windows. Therefore, if the *Routing Problem* is infeasible, the whole problem is infeasible, because there is no possible routing such that tasks are served within their time windows. The information about the previous routes will be stored in *PR* so that each time this algorithm is called, it will provide a new solution to the *Routing Problem*.

If the *Routing Problem* is feasible the next step is to verify whether the available vehicles can execute the routes. The assignment of vehicles to routes is based on the requirements of the routes' tasks for specific types of vehicles, on the routes latest start time, and on the vehicles' operating range and charge rate. This is done by solving the *Assignment Problem*; also in this case there can be multiple feasible solutions, so it is important to store the current one in *CA* to be able to rule it out the next time the *Assignment Problem* is solved. If the *Assignment Problem* is infeasible the algorithm backtracks and runs the *Routing Problem* again, otherwise, it returns a list of feasible assignments *CA* and proceeds to the *Capacity Verification Problem*.

At this point, each route has been assigned an actual vehicle to execute it and start times have been restricted to meet the vehicles' need for charging and the time windows of the tasks they are assigned to. Hence it is possible to verify if the execution of the routes is conflict-free. If that is the case, the

overall problem is feasible and the algorithm terminates and returns a feasible schedule *CVS*. On the other hand, if this step is infeasible, the algorithm will try to find alternative paths for the vehicles to execute the routes.

Table 5.2: Algorithms to solve the sub-problems. Given within parentheses next to the name of the algorithm is the problem that it solves.

Router (*Routing Problem*)
Input: *CP*, *PR* **Output:** *CR* Define optimization problem using (D.5)-(D.19) Optimize and extract *CR* from the solution
Assign (*Assignment Problem*)
Input: *CR*, *PA* **Output:** *CA* Define feasibility problem using (D.20)-(D.25) Solve and extract *CA* from the solution
CapacityVerifier (*Capacity Verification Problem*)
Input: *CA* **Output:** *CVS* Define feasibility problem using (D.26)-(D.32) Solve and extract *CVS* from the solution
PathsChanger (*Paths Changing Problem*)
Input: *PP* **Output:** *NP* Define optimization problem using (D.33)-(D.39) Optimize and extract *NP* from the solution
RoutesVerifier (*Routes Verification Problem*)
Input: *CR*, *NP* **Output:** *True* or *False* Define feasibility problem using (D.40)-(D.43) Solve problem and return *True* if feasible, else *False*

Finding alternative paths is handled in two steps. The *PathsChanger* algorithm finds new paths *NP* and the *RoutesVerifier* makes sure the *Routing Problem* is still solvable (i.e. tasks can still be served within time windows) using these new paths. If the *Paths Changing Problem* is infeasible, all paths

from one task to the next one have been checked for each route. Therefore the algorithm backtracks and looks for a new assignment. Otherwise if the *Paths Changing Problem* is feasible, the algorithm moves forward to the *Routes Verification Problem*. If the *Routes Verification Problem* is feasible, the algorithm backtracks to verify whether the solution using the new paths *NP* is feasible against the capacity constraints by solving the *Capacity Verification Problem*; if not, the *PathsChanger* algorithm is called again.

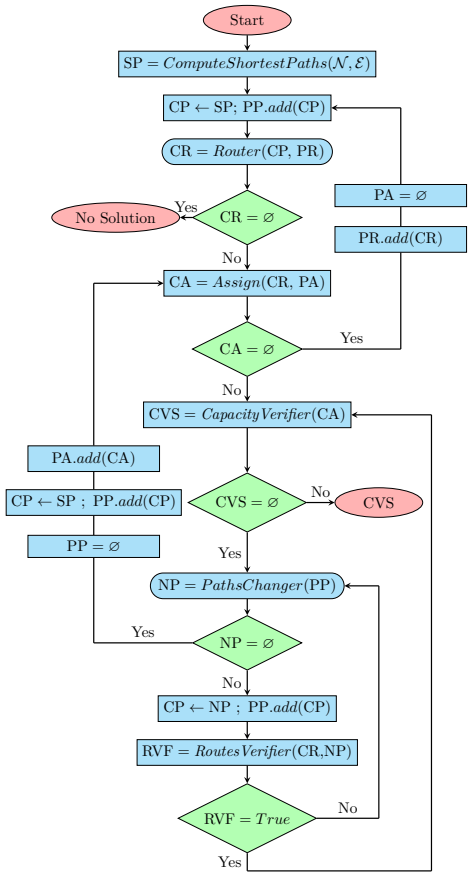

Figure 5.6: Flowchart of ComSat. The *Router* and *PathsChanger* algorithms are put in rounded corner boxes, to remark that they are optimization problems.

Whenever the *Assignment Problem* is infeasible, all possible assignments for the current set of routes *CR* have been explored. Thus, before calling the *Router* algorithm again, *CR* is added to *PR*. In the same way, whenever the *Paths Changing Problem* is infeasible, all possible paths for the current assignment *CA* have been explored, hence *CA* is added to *PA*. Also, the set of previous paths *PP* is emptied because these paths are only eligible for the current assignment, and the shortest paths are set as current paths to compute the next assignment.

On the other hand, when exploring different paths the current assignment is not changed, it is only checked whether it is feasible with the new paths; thus, *CA* is not added to *PA*. Finally, as ComSat loops through *PathsChanger* and *RoutesVerifier* to find a feasible set of paths, *NP* is assigned to *CP*, which in turn is added to *PP* after every unsuccessful iteration.

5.3 Strengths and Weaknesses of ComSat

ComSat has been tested on instances of the CF-EVRP to evaluate its performance. First, a set of smaller problem instances, counting up to four vehicles, seven jobs, a time horizon of 60 time-steps and a graph of 25 nodes, is used to compare the performance of ComSat and MonoMod; then, a set of larger instances counting up to eleven vehicles, fifteen jobs, a time horizon of 300 time-steps and a graph of 35 nodes, is generated to find out how big a problem can be solved in reasonable time by ComSat. In Table 4 of Paper D the results of the first set of experiments are reported. On average, when solving feasible instances, ComSat is faster than MonoMod, with increasing gap as the size of the instances increases. Also, unlike for MonoMod, the performance of ComSat is not affected by the time horizon; this is because breaking down the problem the way we did allows to avoid time discretization, hence the time horizon no longer affects the problem's size. On the other hand, when solving instances that turn out to be infeasible, MonoMod outperforms ComSat in many cases. Further investigations on the instances showed that the bottlenecks of ComSat are *Router* and the *PathChanger* (the next two sections provide additional details on the subject).

In the second set of experiments, ComSat is able to find a solution to most of the instances, or to declare them infeasible within the time limit of 1200 seconds. In quite a number cases it actually shows better performance

compared to the smaller instances from the first set of experiments. This is because an instance can be infeasible for different reasons, and in some cases it takes only a few iterations to determine feasibility/infeasibility. For instance, there could be a limited number of feasible sets of routes that satisfy all the constraints in the *Routing Problem*; if they all turn out to cause infeasibility in one of the next sub-problems, then the whole instance is immediately declared infeasible. However, as shown in Table 5 of Paper D, the solving time increases as the instances become larger and for the larger sets, only a few instances are solved before the time limit.

Testing different formulations of the *Routing Problem*

When it comes to the *Router*, the goal is the computation of feasible routes. The model formulation we used is inspired by the two-index formulation (see Section 3.4). As previously mentioned, this formulation helps to break the symmetry of the problem, an advantage not only when computing one solution, but also when looking for alternative ones. Moreover, for single depot VRPs it is possible to produce feasible solutions using only *two-index variables*. However, for multi depots VRPs, this is not easily achieved; in fact, designing routes that start and end at the same depot (a requirement of the CF-EVRP) would involve a very large number of constraints, that would slow down the solving process significantly. On the other hand, without such constraints, the solver may produce infeasible routes before finding a feasible one.

The alternative is to use a set of three-index binary variables instead (see Section 3.4), where the third index specifies the vehicle that is travelling from customer A to customer B. These variables allow to model the existence of multiple depots and experimental results show that the solving time is actually similar to the solving time for the two-index formulation. The draw back is the symmetry of multiple solutions, since the routes between them could be exactly identical, only executed by different vehicles.

Whether the two-index or three-index formulation is used, as the problem size grows, the number of *Router* solutions that are either infeasible or redundant increases. This does not affect the correctness of the overall solution in the end, because the routes are checked before moving on to the next problem (*Assignment*); however, there is clearly room for improvement in the way this problem is dealt with.

The *Unsat Core* Guided Search for Alternative Paths

Another bottleneck in the solution procedure is the computation of alternative paths. In fact, when the solution using the shortest paths gives rise to conflicts, the *PathChanger* computes an alternative set of paths while trying to minimize their cumulative length. This means that the new set of paths will be very similar to the previous set of paths, which in turn means that probably the conflict will still take place. Experiments on problem instances that were purposefully designed to generate conflicts when using the shortest paths showed that it can take hundreds of calls to the *PathChanger* before a *conflict-free* set of paths is found.

For this reason, in Paper E, a new procedure is presented. This procedure exploits the SMT solver's ability to return a minimal *Unsat Core* and use it to define additional constraints for the *PathChanger* to find conflict-free paths. When the *Capacity Verification Problem* is infeasible, it is possible to query the solver to return one *Unsat Core*. In Paper E we present a procedure to interpret the *Unsat Core* in order to identify the nodes and edges where capacity constraints where violated. This information is then used to formulate additional constraints in the *PathChanger* that forces one of the vehicles involved in the conflict to avoid using the nodes and edges that caused the conflict. Experiments showed that this new approach can solve the conflicts at once, while the previous, more naive approach, would take many iterations.

5.4 A General Framework for Industrial VRPs

In this chapter we saw two methods to solve the CF-EVRP. First we presented MonoMod, the implementation of a monolithic mathematical formulation of the CF-EVRP. The limitations of this approach in terms of performance and the need for a more efficient method led to breaking down the original problem into sub-problems. In Paper D we provided mathematical formulations for each of these sub-problems and a sound and complete procedure, ComSat, to use the sub-problems to find a feasible solution to the CF-EVRP.

Benchmark tests highlighted ComSat's weaknesses. However, ComSat being a compositional algorithm allows to improve single sub-problems' solving procedures without affecting its overall soundness and completeness. In this work we provided proof of concept on the validity of the proposed solution; however, there is much room for improvement in the actual implementation

of the algorithm.

Such implementation may also be problem-dependent. In fact, when facing a specific problem, the end user may find it more convenient to use one strategy rather than another to solve a sub-problem. For instance, when dealing with the *Routing Problem* a local search algorithm [105] may provide faster solutions in some cases, while column generation [106] may be preferred in other cases. This is true for all other sub-problems. In fact, we formulated the sub-problems as either MILP or SMT because modern MILP/SMT solvers are able to quickly solve large models and because, as long as the model fed to the solver is correct, we are sure the solution to the sub-problem is correct too (we assume the algorithms used by the solvers are sound and complete). However, it could be possible to design (or implement existing) tailor made algorithms for each sub-problem that would shorten the computation time significantly.

Moreover, we claim that the CF-EVRP offers a general framework to model several classes of industrial VRPs and, therefore, ComSat is a sound and complete procedure to solve all of them. In fact, VRPs where the capacity of road segments or limited operating range are not considered, where vehicles are homogeneous, or customers have no precedence constraints among each other, etc., are all particular cases of the CF-EVRP.

CHAPTER 6

Summary of included papers

This chapter provides a summary of the included papers. It also puts them into context, discussing the role of each paper and its contribution to the research project that led to this thesis.

The first stage of the research project is the evaluation of SMT and MILP and the comparison of two state-of-the-art SMT and MILP solvers, namely Z3 and Gurobi. This evaluation is presented in papers A and B. In Paper A, the *disjunctive*, *time-indexed*, and *rank-based* models for the standard JSP are modelled as SMT and solved by Z3; the *disjunctive* model turns out to be the fastest. The performance of Gurobi is then compared with Z3 when both solvers are running the *disjunctive* model, and Z3 outperforms Gurobi on each and every problem instance. In Paper B, the concept of *skinny* and *fit* bins for Bin Covering and Bin Packing problems, respectively, is presented. This concept is used to improve on enumerative problem formulations and to develop a heuristic method to quickly obtain sub-optimal solutions. Both the exact and the approximate, heuristic method are formulated as MILP and solved by Gurobi, which proved to be able to handle very large models in reasonable time.

The second stage of the research project is the formalisation of the require-

ments for a material handling system based on AMRs that accounts, among other things, for limited operating range of the robots and capacity constraints on the road segments. This formalization, and the consequent formulation of the CF-EVRP, is presented in Paper C, where the CF-EVRP is introduced, modelled as SMT and solved by Z3. Experiments show that only small problems can be solved in reasonable time.

The results from Paper C show that a monolithic formulation is not suitable to solve large problems in reasonable time. Hence, the next stage is to design a scalable method for the CF-EVRP. Such method is presented in Paper D, where a compositional algorithm (ComSat) to solve the CF-EVRP is presented and compared to the monolithic formulation from Paper C, showing better performance in terms of running time.

Finally, in the last stage of the research project, strengths and weaknesses of ComSat are evaluated and weaknesses are improved upon. The improvements are presented in Paper E, where the search for alternative paths (in case the current ones are conflicting) is handled by exploiting the SMT solvers ability to return *Unsat Cores* and using the information extracted from the *Unsat Core* to guide the search for better paths.

6.1 Paper A

Sabino Francesco Roselli, Kristofer Bengtsson, Knut Åkesson
SMT solvers for job-shop scheduling problems: Models comparison and performance evaluation
Proceedings of 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE),
pp. 547–552, Dec. 2018.
 DOI: 10.1109/COASE.2018.8560344 .

This paper presents a comparison of different model formulations for the standard job shop problem, namely the *disjunctive*, *time-indexed*, and *rank-based* models over a set of generated benchmark problems. The models are implemented for the state-of-the-art SMT solver Z3. Since the *disjunctive* model shows the best performance in terms of solving time, it is selected for further comparison against the state-of-the-art MILP solver Gurobi; Z3 outperforms Gurobi on each and every instance and, in general, is able to solve to optimality medium size problems within 1200 seconds.

6.2 Paper B

Sabino Francesco Roselli, Fredrik Hagebring, Sarmad Riazi, Martin Fabian, Knut Åkesson
On the Use of Equivalence Classes for Optimal and Sub-Optimal Bin Packing and Bin Covering
2021 IEEE Transactions on Automation Science and Engineering (TASE)
vol. 18, no. 1, pp. 369-381, Jan. 2021.
 DOI: 10.1109/TASE.2020.3022986 .

This paper presents a study of the bin sorting problem and how to improve the performance by cutting off portions of the search-space. The improvement is designed for an enumerative formulation, where decision variables represent potential bins; reasoning about the problem constraints allows to eliminate all those potential bins that, given their size (in terms of cumulative value of the items within them), could never be part of an optimal solution. The new approach shows good performance when compared to the standard formulation and can be implemented for both *packing* and *covering* problems. It is supposed to scale very well in terms of number of items, though being based on potential combinations of items, a wide range of values for the items can quickly lead to a state-space explosion. In the paper is also presented a heuristic approach were classes of bins' values are merged together in order to simplify the problem and reduce the solving time, at the cost of solution quality. The heuristic method is compared to the optimal one and proves to significantly reduce the computation time while only slightly reduce the solution quality.

6.3 Paper C

Sabino Francesco Roselli, Martin Fabian, Knut Åkesson
Solving the Conflict-Free Electric Vehicle Routing Problem Using SMT Solvers
Proceedings of 2021 29th Mediterranean Conference on Control and Automation (MED),
pp. 542–547, Dec. 2018.
 DOI: 10.1109/MED51440.2021.9480202 .

This paper introduces the CF-EVRP, the problem of scheduling and routing

of vehicles with given time-windows for delivering material, limited capacity constraints of the road network, and with the need for battery recharge. The problem is formulated as SMT and evaluated on a set of generated problem instances using the state-of-the-art SMT solver Z3. Experiments show that the solver can handle medium size instances in a reasonable amount of time.

6.4 Paper D

Sabino Francesco Roselli, Per-Lage Götvall, Martin Fabian, Knut Åkesson
A Compositional Algorithm for the Conflict-Free Electric Vehicle Routing Problem
2022 IEEE Transactions on Automation Science and Engineering (TASE)
pp. 1405 - 1421, May. 2022.
 DOI: 10.1109/TASE.2022.3169949 .

In this paper, the compositional algorithm ComSat for solving the CF-EVRP is presented. The CF-EVRP is decomposed into sub-problems and ComSat iterates through them until a globally feasible solution is found. The proposed algorithm is implemented using an optimizing SMT-solver and is evaluated against an implementation of a previously presented monolithic model. The soundness and completeness of the algorithm are proven, and it is benchmarked on a set of generated problems and found to be able to solve problems of industrial size.

6.5 Paper E

Sabino Francesco Roselli, Remco Vader, Martin Fabian, Knut Åkesson
Leveraging Conflicting Constraints in Solving Vehicle Routing Problems
Proceedings of 16th IFAC Workshop on Discrete Event Systems (WODES) September 7-9, 2022 - Prague, Czechia .

Previous work introduced the compositional algorithm *ComSat* that solves the CF-EVRP by searching a strongly connected, weighted, directed graph modeling the problem. Though ComSat showed good performance in general, some problems took unreasonably long time to solve due to the high number of iterations required to find solutions with respect to the road segments' capacity

constraints. The bottleneck is the *Path Changing Problem*, i.e., the problem of finding a new set of shortest paths to connect a subset of the customers, disregarding previously found shortest paths. This paper presents an improved version of the *PathsChanger* function to solve the *Path Changing Problem* that exploits the *unsatisfiable core*, i.e., information of which constraints conflict, to guide the search for feasible solutions. Experiments show faster convergence to feasible solutions compared to the previous version of *PathsChanger*.

CHAPTER 7

Concluding Remarks and Future Work

In this thesis we have discussed the challenges of modern industrial problems in general, and of material handling by means of transportation robots in particular. We have presented the CF-EVRP as a novel problem in the field, whose formulation is motivated by the need of modelling industrial scenarios where mobile robots operate together with human workers and other vehicles, and have to be able to react to an ever changing environment. The CF-EVRP offers a general framework that can be used to model several industrial scenarios involving vehicles with limited operating range, limited capacity of the plant layout in terms of vehicles that can access certain road segments at the same time, time windows for delivery, and more.

We analyzed the scalability of the CF-EVRP by formulating it as SMT and solved it using the state-of-the-art SMT solver Z3. The experiments showed the limitations of using a monolithic model, MonoMod, to solve the problem in terms of computation time, hence we developed the compositional algorithm, ComSat, to be able to handle larger problem instances. As a first step, we decomposed the CF-EVRP into sub-problems and provided mathematical formulations for each of them. Then we designed ComSat to iterate through these sub-problems to find a feasible solution to the CF-EVRP.

In our implementation of ComSat, the sub-problems are solved by means of general purpose solvers, i.e., either MILP or SMT. In order to choose the best solver for each sub-problem we tested their performance on well-known optimization problems.

Experiments on the JSP showed Z3 to be significantly faster than state-of-the-art MILP solver Gurobi, while for the BSP and the VRPTW it was the other way around. Later investigations showed that the JSP has a structure that can be efficiently exploited by Z3, since its constraints fall into the category of *difference logic*, a fragment of linear arithmetic for which there exist polynomial algorithms. Z3 can use a *lazy approach* (see Chapter 3) to find assignments for such constraints and check their consistency using one of these polynomial algorithms. As for BSP and VRPTW there is no particular structure that Z3 can exploit and the superiority of Gurobi in handling linear arithmetic is evident.

Based on these findings we formulated some sub-problems of the CF-EVRP as JSP and used Z3 to solve them in ComSat. We applied the same reasoning for the other sub-problems, choosing the solver that would better exploit the sub-problem structure. We designed a set of benchmark instances to test ComSat and MonoMod and compare their performance. These tests showed a significantly better scalability of ComSat compared to MonoMod, but also highlighted the compositional algorithm bottlenecks. One bottleneck is the search for alternative paths when the shortest ones cause conflicts.

We therefore exploited the concept of *Unsat Core* and the ability of Z3 to return *Unsat Cores* when a problem is infeasible to develop an improved method (part of ComSat) for the search of alternative paths. Preliminary tests on this method showed a much lower number of iterations needed to find *conflict-free* paths.

In general, based on our experiments and finding, we here answer the research questions we formulated in Chapter 1:

RQ1 What are the strengths and weaknesses of SMT solvers when used to solve industrial problems?

We formulated all the problems presented in this thesis using both MILP and SMT, except for MonoMod (only implemented as SMT), although that would be possible too. From a modelling perspective, SMT solvers allow for logical constraints over literals of different theories, including combinations of

them; when using MILP solvers instead, only conjunctions of inequalities over reals and integers are allowed, therefore modelling may be trickier, since it requires declaring additional variables and modelling techniques such as the *big M* method [107].

For instance, when modelling the JSP, the non-overlapping of operations sharing the same resource can be directly modelled with an SMT solver as a disjunction of two linear inequalities. On the other hand, modelling the non-overlapping constraint with MILP is only possible with the support of an auxiliary binary variable and a large enough constant (the *big M* method), as shown in Section 3.3.

When it comes to performance, SMT solvers are preferable when the problem structure involves complex logical constraints such as the disjunctive constraints for the JSP, as discussed in Paper A. On the other hand, MILP solvers are faster if the problem mainly involves conjunctions of inequalities; an example of this is the equivalence class method presented in Paper B. The method involves a linear model with only conjunctions of inequalities over integer variables; the comparison of Z3 and Gurobi for problems formulated using the equivalence class method showed that Gurobi outperformed Z3.

RQ2 How can the strengths of SMT and/or MILP solvers be exploited and combined to design an efficient algorithm for the CF-EVRP?

ComSat takes advantage of the CF-EVRP decomposition into sub-problems to incrementally build an overall solution based on each sub-problem solution. When formulating the sub-problems, the *Assignment* and *CapacityVerification* problems are modelled as variants of the JSP and solved using Z3, since computational analysis presented in Paper A shows the superiority of SMT over MILP when solving a JSP. Also the *PathChanger* has been modelled using Boolean variables and logical constraints, a formulation that fits an SMT solver better than a MILP solver. Moreover, in Paper E the search for alternative paths has been improved using *Unsat Cores* extraction, a feature available for Z3 but not Gurobi. On the other hand, the *Routing Problem* is a variant of a VRPTW, for which MILP showed better performance than SMT, hence we formulated it and solved it using Gurobi.

7.1 Future Work

One of the goals of this research project was to create a general framework to formulate (CF-EVRP) and solve (ComSat) a wide range of transportation problems in industrial plants. This goal has partially been achieved, since it is possible to model systems with the following features:

- heterogeneous fleets of AMRs, with limited operating range and non-negligible charging times;
- jobs with specific requirements in terms of time windows, eligible vehicles, precedence constraints, and service time;
- plants having having multiple depots and capacity constraints on their driving roads in terms of the number of vehicles that can be accommodated simultaneously.

However, it is relevant to extend the current formulation to handle the additional features and remove restrictions that were defined in order to develop ComSat:

- it is not possible to select more than one vehicle to execute a task;
- vehicles travel on closed routes, i.e,. they have to start and end at the same depot;
- the paths computed for the vehicles to travel between customers cannot involve cycles, a feature that can potentially remove feasible solutions, compared to a real system.

Moreover, though ComSat shows promising performance when solving instances from the generated benchmark set, there is room for improvement in terms of implementation that would allow it to scale to even larger problems. For instance, it would be possible to work on each sub-problem separately and develop a tailor made algorithm for each of them, rather than using general purpose solvers. Since each sub-problem is modelled as a variant of a well-known optimization problem there is plenty of literature to take inspiration from. Clustering [108] could help splitting one instance of the CF-EVRP into several smaller ones, speeding up the computation process at the cost of the solution quality. Also worth mentioning, the version of ComSat presented in

Paper D is proven to be sound and complete. However, it has been shown how, for some instances, in order to guarantee that the answer returned is correct, a lot of iterations are needed. Maybe some heuristic procedure could be introduced in the algorithm that trade soundness and completeness for better performance.

Finally, when comparing the quality of the solutions yielded by MonoMod and ComSat, we used the total travelled distance as objective function. We did so because it is a standard parameter to optimize in VRPs. However, the total travelled distance may not be the best option for an industrial application. Other possibilities could be the minimization of the make-span, or the number of vehicles out in the plant at the same time. Changing the objective function could produce much different solutions, and also change the computation time significantly, a feature that needs being investigated.

Future work in this research project would be to improve on the above mentioned features and embed them into an online framework that allows to recompute a new schedule when changes in the plant require it. For this purpose, a variation of ComSat based on a rolling horizon rather than a fixed-time horizon may better suit industrial problems, having to deal with only a short time span, and also reflecting better the ever changing environment of an industrial plant.

References

[1] K. H. Tantawi, A. Sokolov, and O. Tantawi, "Advances in industrial robotics: From industry 3.0 automation to industry 4.0 collaboration," in *2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON)*, IEEE, 2019, pp. 1–4.

[2] C. Audet and W. Hare, *Derivative-free and blackbox optimization*. Springer, 2017, ISBN: 978-3-319-68912-8.

[3] A. P. Barbosa-Povoa and J. M. Pinto, "Process supply chains: Perspectives from academia and industry," *Computers & Chemical Engineering*, vol. 132, 2020.

[4] M. L. Varela, G. D. Putnik, V. K. Manupati, G. Rajyalakshmi, J. Trojanowska, and J. Machado, "Integrated process planning and scheduling in networked manufacturing systems for I4.0: A review and framework proposal," *Wireless Networks*, vol. 27, no. 3, pp. 1587–1599, 2021.

[5] J. Kallrath, "Planning and scheduling in the process industry," *OR spectrum*, vol. 24, no. 3, pp. 219–250, 2002.

[6] D. P. Williamson and D. B. Shmoys, *The design of approximation algorithms*. Cambridge university press, 2011.

[7] M. Guignard and S. Kim, "Lagrangean decomposition for integer programming: Theory and applications," *RAIRO-Operations Research*, vol. 21, no. 4, pp. 307–323, 1987.

[8] G. Desaulniers, J. Desrosiers, and M. M. Solomon, *Column generation*. Springer Science & Business Media, 2006, vol. 5.

[9] R. Rahmaniani, T. G. Crainic, M. Gendreau, and W. Rei, "The Benders decomposition algorithm: A literature review," *European Journal of Operational Research*, vol. 259, no. 3, pp. 801–817, 2017.

[10] E. Kondili, C. C. Pantelides, and R. W. Sargent, "A general algorithm for short-term scheduling of batch operations—I. MILP formulation," *Computers & Chemical Engineering*, vol. 17, no. 2, pp. 211–227, 1993.

[11] C. Barrett and C. Tinelli, "Satisfiability modulo theories," *Handbook of model checking*, pp. 305–343, 2018.

[12] S. Wang, X. Liao, M. Wang, L. Chang, H. Yang, and T. Wang, "An improved SMT-based scheduling for overloaded real-time systems," *Engineering Letters*, vol. 28, no. 1, 2020.

[13] M. Hekmatnejad, G. Pedrielli, and G. Fainekos, "Task scheduling with nonlinear costs using SMT solvers," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2019, pp. 183–188.

[14] T. Weber, S. Conchon, D. Déharbe, M. Heizmann, A. Niemetz, and G. Reger, "The SMT competition 2015–2018," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 11, no. 1, pp. 221–259, 2019.

[15] T. Berthold, T. Koch, and Y. Shinano, "MILP. Try. Repeat.," in *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling-PATAT 2021*, vol. 2, 2021.

[16] L. Meng, K. Gao, Y. Ren, B. Zhang, H. Sang, and Z. Chaoyong, "Novel MILP and CP models for distributed hybrid flowshop scheduling problem with sequence-dependent setup times," *Swarm and Evolutionary Computation*, vol. 71, 2022.

[17] A. I. Corréa, A. Langevin, and L.-M. Rousseau, "Scheduling and routing of automated guided vehicles: A hybrid approach," *Computers & operations research*, vol. 34, no. 6, pp. 1688–1707, 2007.

[18] H. Li and K. Womer, "Scheduling projects with multi-skilled personnel by a hybrid MILP/CP Benders decomposition algorithm," *Journal of Scheduling*, vol. 12, no. 3, pp. 281–298, 2009.

[19] V. Jain and I. E. Grossmann, "Algorithms for hybrid MILP/CP models for a class of optimization problems," *INFORMS Journal on computing*, vol. 13, no. 4, pp. 258–276, 2001.

[20] J. Roslöf, I. Harjunkoski, T. Westerlund, and J. Isaksson, "Solving a large-scale industrial scheduling problem using MILP combined with a heuristic procedure," *European Journal of Operational Research*, vol. 138, no. 1, pp. 29–42, 2002.

[21] K. Azadeh, M. deKoster, and D. Roy, "Robotized warehouse systems: Developments and research opportunities," *ERIM Report Series Research in Management*, no. ERS-2017-009-LIS, 2017.

[22] A. G. Frank, L. S. Dalenogare, and N. F. Ayala, "Industry 4.0 technologies: Implementation patterns in manufacturing companies," *International Journal of Production Economics*, vol. 210, pp. 15–26, 2019.

[23] S. Roselli, M. Fabian, and K. Åkesson, "Solving the Electric-Conflict Free-Vehicle Routing Problem Using SMT Solvers," *2021 29th Mediterranean Conference on Control and Automation (MED)*, pp. 542–547, 2021.

[24] A. S. Manne, "On the job-shop scheduling problem," *Operations Research*, vol. 8, no. 2, pp. 219–223, 1960.

[25] M. G. S. Furtado, P. Munari, and R. Morabito, "Pickup and delivery problem with time windows: A new compact two-index formulation," *Operations Research Letters*, vol. 45, no. 4, pp. 334–341, 2017.

[26] L. V. Kantorovich, "Mathematical methods of organizing and planning production," *Management science*, vol. 6, no. 4, pp. 366–422, 1960.

[27] J. E. Hopcroft and J. D. Ullman, *Formal languages and their relation to automata*. Addison-Wesley Longman Publishing Co., Inc., 1969.

[28] J. Kleinberg and E. Tardos, *Algorithm design*. Pearson Education India, 2006.

[29] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*, Springer, 1972, pp. 85–103.

[30] D. E. Knuth, *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.

[31] L. J. Stockmeyer, "Computational complexity," *Handbooks in Operations Research and Management Science*, vol. 3, pp. 455–517, 1992.

[32] M. Bellare and S. Goldwasser, "The complexity of decision versus search," *SIAM Journal on Computing*, vol. 23, no. 1, pp. 97–119, 1994.

[33] J. Van Leeuwen, *Handbook of theoretical computer science (vol. A) algorithms and complexity*. Mit Press, 1991.

[34] H. Fisher, "Probabilistic learning combinations of local job-shop scheduling rules," *Industrial scheduling*, pp. 225–251, 1963.

[35] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, May 1976, ISSN: 0364-765X.

[36] A. S. Jain and S. Meeran, "Deterministic job-shop scheduling: Past, present and future," *European Journal of Operational Research*, vol. 113, no. 2, pp. 390–434, 1999.

[37] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.

[38] S. Lawrence, "Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement)," *Graduate School of Industrial Administration, Carnegie-Mellon University*, 1984.

[39] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[40] R. H. Storer, S. D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Management science*, vol. 38, no. 10, pp. 1495–1509, 1992.

[41] I. A. Chaudhry and A. A. Khan, "A research survey: Review of flexible job shop scheduling techniques," *International Transactions in Operational Research*, vol. 23, no. 3, pp. 551–591, 2016.

[42] A. Mascis and D. Pacciarelli, "Job-shop scheduling with blocking and no-wait constraints," *European Journal of Operational Research*, vol. 143, no. 3, pp. 498–517, 2002.

[43] K. Jansen, S. Kratsch, D. Marx, and I. Schlotter, "Bin packing with fixed number of bins revisited," *Journal of Computer and System Sciences*, vol. 79, no. 1, pp. 39–49, 2013.

[44] E. G. Coffman, J. Csirik, G. Galambos, S. Martello, and D. Vigo, "Bin packing approximation algorithms: Survey and classification," in *Handbook of combinatorial optimization*, Springer New York, 2013, pp. 455–531.

[45] M. Delorme, M. Iori, and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," *European Journal of Operational Research*, vol. 255, no. 1, pp. 1–20, 2016.

[46] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *Journal of heuristics*, vol. 2, no. 1, pp. 5–30, 1996.

[47] A. Scholl, R. Klein, and C. Jürgens, "Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem," *Computers & Operations Research*, vol. 24, no. 7, pp. 627–645, 1997.

[48] P. Schwerin and G. Wäscher, "The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP," *International Transactions in Operational Research*, vol. 4, no. 5-6, pp. 377–389, 1997.

[49] G. Wäscher and T. Gau, "Heuristics for the integer one-dimensional cutting stock problem: A computational study," *Operations-Research-Spektrum*, vol. 18, no. 3, pp. 131–144, 1996.

[50] J. E. Schoenfield, "Fast, exact solution of open bin packing problems without linear programming. draft," *US Army Space & Missile Defence Command, Huntsville*, vol. 20, p. 72, 2002.

[51] M. R. Garey and D. S. Johnson, "Approximation algorithms for bin packing problems: A survey," in *Analysis and design of algorithms in combinatorial optimization*, Springer, 1981, pp. 147–172.

[52] M. G. Christ, L. M. Favrholdt, and K. S. Larsen, "Online bin covering: Expectations vs. guarantees," *Theoretical Computer Science*, vol. 556, pp. 71–84, 2014.

[53] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[54] J. K. Lenstra and A. R. Kan, "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, no. 2, pp. 221–227, 1981.

[55] T. K. Ralphs, L. Kopman, W. R. Pulleyblank, and L. E. Trotter, "On the capacitated vehicle routing problem," *Mathematical programming*, vol. 94, no. 2, pp. 343–359, 2003.

[56] M. Schneider, A. Stenger, and D. Goeke, "The electric vehicle-routing problem with time windows and recharging stations," *Transportation Science*, vol. 48, no. 4, pp. 500–520, 2014.

[57] M. Desrochers, J. Desrosiers, and M. Solomon, "A new optimization algorithm for the vehicle routing problem with time windows," *Operations research*, vol. 40, no. 2, pp. 342–354, 1992.

[58] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations research*, vol. 35, no. 2, pp. 254–265, 1987.

[59] H. Zhang, H. Ge, J. Yang, and Y. Tong, "Review of vehicle routing problems: Models, classification and solving algorithms," *Archives of Computational Methods in Engineering*, pp. 1–27, 2021.

[60] J. Lundgren, M. Rönnqvist, and P. Värbrand, *Optimization*. Lund, Sweden: Studentlitteratur, 2010, ISBN: 9789144053080.

[61] D. Kroening and O. Strichman, *Decision procedures - An Algorithmic Point of View*. Springer, 2016.

[62] G. B. Dantzig, "Origins of the simplex method," in *A history of scientific computing*, 1990, pp. 141–151.

[63] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, 1984, pp. 302–311.

[64] M. R. Lambrecht, P. L. Ivens, and N. J. Vandaele, "ACLIPS: A capacity and lead time integrated procedure for scheduling," *Management Science*, vol. 44, no. 11-part-1, pp. 1548–1561, 1998.

[65] M. Sipser, "Introduction to the theory of computation," *ACM Sigact News*, vol. 27, no. 1, pp. 27–29, 1996.

[66] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik, "Efficient conflict driven learning in a boolean satisfiability solver," in *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No. 01CH37281)*, IEEE, 2001, pp. 279–285.

[67] J. Franco and J. Martin, "A history of satisfiability.," *Handbook of satisfiability*, vol. 185, pp. 3–74, 2009.

[68] M. J. Heule, M. Järvisalo, and M. Suda, "SAT competition 2018," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 11, no. 1, pp. 133–154, 2019.

[69] G. Tseitin, "On the complexity of proofs in propositional logics," in *Seminars in Mathematics*, vol. 8, 1970, pp. 466–483.

[70] L. Hadarean, K. Bansal, D. Jovanović, C. Barrett, and C. Tinelli, "A tale of two solvers: Eager and lazy approaches to bit-vectors," in *International Conference on Computer Aided Verification*, Springer, 2014, pp. 680–695.

[71] R. Bruttomesso, "An extension of the Davis-Putnam procedure and its application to preprocessing in SMT," in *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, 2009, pp. 14–19.

[72] R. Nieuwenhuis and A. Oliveras, "On SAT modulo theories and optimization problems," in *International conference on theory and applications of satisfiability testing*, Springer, 2006, pp. 156–169.

[73] N. Bjørner and A.-D. Phan, "νZ-maximal satisfaction with Z3.," *SCSS*, vol. 30, pp. 1–9, 2014.

[74] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νZ-an optimizing SMT solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2015, pp. 194–199.

[75] B. Dutertre and L. De Moura, "A fast linear-arithmetic solver for DPLL (T)," in *International Conference on Computer Aided Verification*, Springer, 2006, pp. 81–94.

[76] R. Sebastiani and P. Trentin, "OptiMathSAT: A tool for optimization modulo theories," in *International Conference on Computer Aided Verification*, Springer, 2015, pp. 447–454.

[77] I. Lynce and J. P. Marques-Silva, "On computing minimum unsatisfiable cores," in *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing*, Vancouver, Canada, 2004, pp. 305–310.

[78] J. P. Tremblay and R. Manohar, *Discrete mathematical structures with applications to computer science*. McGraw-Hill, Inc., 1975.

[79] Gurobi Optimization, LLC, *Gurobi optimizer reference manual, version 8.1*, 2019.

[80] W.-Y. Ku and J. C. Beck, "Mixed integer programming models for job shop scheduling: A computational analysis," *Computers & Operations Research*, vol. 73, pp. 165–173, 2016.

[81] R. Lima, "IBM ILOG CPLEX–What is inside of the box," in *Proc. 2010 EWO Seminar*, 2010, pp. 1–72.

[82] R. Sebastiani and P. Trentin, "OptiMathSAT: A tool for optimization modulo theories," *Journal of Automated Reasoning*, vol. 64, no. 3, pp. 423–460, 2020.

[83] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[84] J.-F. Cordeau, G. Desaulniers, J. Dsrosiers, M. M. Solomon, and F. Soumis, *The VRP with time windows*. Groupe d'études et de recherche en analyse des décisions Montréal, 2000.

[85] P. Toth and D. Vigo, "VRP with backhauls," in *The vehicle routing problem*, SIAM, 2002, pp. 195–224.

[86] F. A. Tillman, "The multiple terminal delivery problem with probabilistic demands," *Transportation Science*, vol. 3, no. 3, pp. 192–204, 1969.

[87] J. Li, Y. Li, and P. M. Pardalos, "Multi-depot vehicle routing problem with time windows under shared depot resources," *Journal of combinatorial optimization*, vol. 31, no. 2, pp. 515–532, 2016.

[88] J.-F. Cordeau, M. Gendreau, and G. Laporte, "A tabu search heuristic for periodic and multi-depot vehicle routing problems," *Networks: An International Journal*, vol. 30, no. 2, pp. 105–119, 1997.

[89] Ç. Koç, T. Bektaş, O. Jabali, and G. Laporte, "Thirty years of heterogeneous vehicle routing," *European Journal of Operational Research*, vol. 249, no. 1, pp. 1–21, 2016.

[90] J. F. Bard, L. Huang, M. Dror, and P. Jaillet, "A branch and cut algorithm for the VRP with satellite facilities," *IIE Transactions*, vol. 30, no. 9, pp. 821–834, 1998.

[91] S. Erdoğan and E. Miller-Hooks, "A green vehicle routing problem," *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 1, pp. 100–114, 2012.

[92] J. Lin, W. Zhou, and O. Wolfson, "Electric vehicle routing problem," *Transportation Research Procedia*, vol. 12, no. Supplement C, pp. 508–521, 2016.

[93] G. Hiermann, J. Puchinger, S. Ropke, and R. F. Hartl, "The electric fleet size and mix vehicle routing problem with time windows and recharging stations," *European Journal of Operational Research*, vol. 252, no. 3, pp. 995–1018, 2016.

[94] A. Wallar, J. Alonso-Mora, and D. Rus, "Optimizing vehicle distributions and fleet sizes for shared mobility-on-demand," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 3853–3859.

[95] M. Salazar, F. Rossi, M. Schiffer, C. H. Onder, and M. Pavone, "On the interaction between autonomous mobility-on-demand and public transportation systems," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2018, pp. 2262–2269.

[96] Y. Liu, S. Ji, Z. Su, and D. Guo, "Multi-objective AGV scheduling in an automatic sorting system of an unmanned (intelligent) warehouse by using two adaptive genetic algorithms and a multi-adaptive genetic algorithm," *PloS one*, vol. 14, no. 12, 2019.

[97] N. N. Krishnamurthy, R. Batta, and M. H. Karwan, "Developing conflict-free routes for automated guided vehicles," *Operations Research*, vol. 41, no. 6, pp. 1077–1090, 1993.

[98] M. Saidi-Mehrabad, S. Dehnavi-Arani, F. Evazabadian, and V. Mahmoodian, "An ant colony algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs," *Computers & Industrial Engineering*, vol. 86, pp. 2–13, 2015.

[99] R. Yuan, T. Dong, and J. Li, "Research on the collision-free path planning of multi-AGVs system based on improved A* algorithm," *American Journal of Operations Research*, vol. 6, no. 6, pp. 442–449, 2016.

[100] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[101] E. Thanos, T. Wauters, and G. Vanden Berghe, "Dispatch and conflict-free routing of capacitated vehicles with storage stack allocation," *Journal of the Operational Research Society*, pp. 1–14, 2019.

[102] K. Murakami, "Time-space network model and MILP formulation of the conflict-free routing problem of a capacitated AGV system," *Computers & Industrial Engineering*, 2020.

[103] M. Zhong, Y. Yang, Y. Dessouky, and O. Postolache, "Multi-AGV scheduling for conflict-free path planning in automated container terminals," *Computers & Industrial Engineering*, vol. 142, 2020.

[104] F. A. Aloul, B. Al Rawi, and M. Aboelaze, "Identifying the shortest path in large networks using boolean satisfiability," in *2006 3rd International Conference on Electrical and Electronics Engineering*, IEEE, 2006, pp. 1–4.

[105] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, part I: Route construction and local search algorithms," *Transportation science*, vol. 39, no. 1, pp. 104–118, 2005.

[106] B. Kallehauge, J. Larsen, O. B. Madsen, and M. M. Solomon, "Vehicle routing problem with time windows," in *Column generation*, Springer, 2005, pp. 67–98.

[107] F. Trespalacios and I. E. Grossmann, "Improved big-M reformulation for generalized disjunctive programs," *Computers & Chemical Engineering*, vol. 76, pp. 98–103, 2015.

[108] M. G. Omran, A. P. Engelbrecht, and A. Salman, "An overview of clustering methods," *Intelligent Data Analysis*, vol. 11, no. 6, pp. 583–605, 2007.

Part II

Papers

PAPER A

SMT solvers for job-shop scheduling problems: Models comparison and performance evaluation

Sabino Francesco Roselli, Kristofer Bengtsson, Knut Åkesson

Proceedings of 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE),
pp. 547–552, Dec. 2018.
 DOI: 10.1109/COASE.2018.8560344

The layout has been revised.

Abstract

The optimal assignment of jobs to machines is a common problem when implementing automated production systems. A specific variant of this category is the job-shop scheduling problem (JSP) that is known to belong to the class of NP-hard problems. JSPs are typically either formulated as Mixed Integer Linear Programming (MILP) problems and solved by general-purpose-MILP solvers or approached using heuristic algorithms specifically designed for the purpose.
During the last decade a new approach, satisfiability (SAT), led to develop solvers with incredible abilities in finding feasible solutions for hard combinatorial problems on Boolean variables. Moreover, an extension of SAT, Satisfability Modulo Theory (SMT), allows to formulate constraints involving linear operations over integers and reals and some SMT-solvers have been also extended with an optimizing tool.
Since the JSP is a well-known hard combinatorial problem, it is interesting to evaluate how SMT-solvers perform in solving it and how they compare to traditional MILP-solvers. We therefore evaluate state-of-the-art MILP and SMT solvers on benchmark JSP instances and find that general-purpose open-source SMT-solvers are competitive against commercial MILP-solvers.

1 Introduction

The Job-shop scheduling problem (JSP) is a well known problem within the Operation Scheduling Community, where the target is to allocate resources to operations while minimizing some cost function. In [1] a thorough study on the subject is presented, providing information about the evolution of techniques and algorithms to deal with the JSP whether the target was the true optimal or an approximation of it.

Industrial scheduling problems are typically extensions of the pure job-shop problem described above. Additional extensions are the possibility to have alternative resources that can process an operation, constraints on setup-time

for resources, constraints on the idle-time between operations, etc. In many applications it might be desired to not minimize the make-span but instead make scheduling decisions based on given deadlines for operations and then minimize the tardiness or maximum lateness.

A recent study by [2] shows promising results in solving JSP problems by employing Constraint Programming (CP) to implement Local-Search (LS) algorithms. A similar solution has also been implemented by Beck et Al. [3], leading again to good results. Apparently the combination of CP and LS is a powerful instrument to takle the JSP, although also other techniques are used in practice. In fact, according to Beck himself, as shown in [4], both in industry and academia, Mixed Integer Linear Programming (MILP) is largely employed for the JSP. Based on this the authors did a benchmark on three popular MILP solvers, IBM ILOG CPLEX [5], Gurobi [6], and SCIP [7]. CPLEX and Gurobi are considered to be state-of-the-art commercial solvers [8], while SCIP is a fast non-commercial solver. In [4] the authors also compared alternative MILP problem formulations that have been proposed in the literature for the classical job-shop problem. The tools were evaluated on a large set of benchmark job-shop problems. The result of the benchmark is that for the tools evaluated, the performance of CPLEX and Gurobi were comparable and significantly better than SCIP and that the traditional disjunctive formulation of the job-shop problem was superior to both Time-Index and Rank-Based formulations.

While many industrial problems can be solved by general purpose MILP-solvers, the combinatorial complexity of the JSP implies that for sufficiently large problems it will not be possible to find an optimal solution within reasonable time. For such problems, specific-purpose methods have been developed: they often make use of heuristics as well as hybrid techniques including local search and genetic algorithms, e.g. [9] and [10]), that lead to *good enough* solutions in a reasonable amount of time [11].

An alternative technology to solve combinatorial problems have emerged within the community of formal verification of hardware and software: satisfability search (SAT) [12]. This approach consists of expressing the problem through Boolean variables in Conjunctive Normal Form (CNF) and determine whether there exists an assignment to these variables such that the formula evaluates to true. Today SAT-solvers can deal with large combinatorial problems by efficiently exploiting their structure and generate a valid model in a reasonable time, even when the problem happens to be NP-complete. Of

course this does not mean that there are not NP-complete problems that will take exponential time also for SAT-solvers, but that many man-made models are no longer intractable.

On the other hand, Boolean logic is in many cases not expressive enough for representing many real-world problems where first-order logic is used together with integers and reals. To handle this type of models and at the same time take advantage of the very performant SAT-solvers, a new approach, called Satisfability Modulo Theory (SMT), [13], [14] has been developed. SMT-solvers implements special decision procedures, so called theories, to extend to the original Boolean satisifiability problem.

Both SAT and SMT are employed to prove satisfability of formulas; it is often the case that not only one but several satisfable solutions exist for a given formula and each of them has a cost referring to some parameter related to the problem itself. It is therefore possible to turn the satisfability problem into an optimization one by setting an objective function where one not only wants to find a satisfable solution, but the optimal one in respect of the cost we assign to the variables.

Ever since SAT and SMT solvers started spreading and being employed in real-world applications, users have built their own custom loops to achieve optimality, often based on heuristics tuned for their specific problems. Lately, some of these solvers have been extended by including optimizing tools that make use of state-of-the-art algorithms to deal with a list of different problems on SMT formulas with linear objective functions on Boolean, rational and integer domain (or a combination of them).

The performance of SMT-solvers are evaluated annually in a benchmark competition. The latest competition was the The 12th International Satisfiability Modulo Theories Competition (SMT-COMP 2017). The SMT-solvers compete in different categories in which Z3 [15], MathSAT5 [16], and CVC4 [17] have shown consistently good performance. Among these solvers Z3 and MathSat5 have versions that support optimization as well. The optimizing version of Z3 is described in [18] and is included in the latest version of Z3. MathSat5 optimizing tool is a special version of it named OptiMathSAT [19]. CVC4 does not have support for optimization and is thus excluded from this benchmark.

In order to have a valid comparison with the current technology, we also tested the models on a MILP solver. We chose Gurobi, since it is considered

to be among the state-of-the-art solvers in such field.

The contributions in this paper are. (i) Adapting three existing formulations of the classical job-shop problem to make them suitable for SMT-solvers. (ii) Benchmarking two optimizing SMT-solvers on a set of benchmark job-shop problems.

In the next section the problem is described in detail, providing all necessary information to understand the models. In section three, we describe Z3 implementation for such models, we compare Z3 performance with Gurobi and OptiMathSAT using the Disjunctive model and we discuss the results. Finally we draw conclusions in section four.

2 Problem Description

The JSP problem consists of a set of n jobs $J = \{j_i\}_{i=1}^{n}$, where each job has its own processing order through a set of k machines, $M = \{m_i\}_{i=1}^{k}$. Operations are defined as the execution of a job on a certain machine and, as each job has to visit each machine, the total number of operations in the problem is $n \cdot k$. Each job will go through all machines sequentially. Let o_i^j model the index of the machine to be used for job j executing operation i in sequence. The index of the machines for each step in the job sequence is thus given by $(o_1^j, \ldots, o_i^j, \ldots, o_k^j)$. Also, let d_i^j model the duration of the execution of the same operation.

Finding a solution to the job-shop scheduling problem means to assign operations to machines so that all jobs are completed. The constraints in this kind of problem are two:

- as there exist a sequence of operations for each job, operations belonging to the same job must be executed in the right order;
- operations requiring the same machine and belonging to different jobs cannot overlap in time.

Given these two constraints, the target is to find a feasible schedule such that the overall make-span is minimized. Finding an optimal schedule is an NP-complete problem, [20].

We now present the three model formulations, (i) the Disjunctive model, (ii) the Time-Index model, and (iii) the Rank-Based model for the classical JSP, based on [4]. They are adapted to fit the SMT language.

2.1 Time-Index Model

In this model the execution time is split into steps, whose length is the minimum time unit. For instance, if the duration of an operation is n-seconds (or minutes), n steps will be taken since it starts and until it is completed. In order to create a model with such feature, we need to have a guess of the overall execution time, in other words, an upper bound H. A trivial upper bound is the sum of all operations duration as if they were executed in a sequence (the worst case scenario). The greater the upper bound, the longer it takes to create the model and therefore the slower the overall execution. Nevertheless, as finding good upper bounds for such model is beyond the scope of this paper, the trivial one is used. In this model the decision variable is:

- s_{mjt} is a Boolean variable that is true if job j starts on machine m at time t.

minimize T_{max} subject to

$$\bigvee_{t=0}^{H} s_{mjt} \qquad \forall m \in M, j \in J \tag{A.1}$$

$$s_{mjt} \rightarrow \bigwedge_{t'=0}^{t-1} \neg s_{mjt'} \wedge \bigwedge_{t'=t+1}^{H} \neg s_{mjt'}$$
$$\forall t = 1, \ldots, H, m \in M, j \in J \tag{A.2}$$

$$s_{mjt} \rightarrow \bigwedge_{t'=t}^{t+p_{mj}} \neg s_{mj't'}$$
$$\forall j, j' \in J, j \leq j', t = 1, \ldots, H, m \in M \tag{A.3}$$

$$s_{mjt} \rightarrow T_{max} \geq t + d_{mj}$$
$$\forall m \in M, j \in J, t = 1, \ldots, H \tag{A.4}$$

$$x_{o^j_{i-1}jt} \rightarrow \bigwedge_{t'=0}^{t+d^j_{i-1}} \neg x_{o^j_{i-1}jt'}$$
$$\forall i = 2, \ldots, k, t = 1, \ldots, H, j \in J \tag{A.5}$$

Equation (A.1) and (A.2) are needed to make sure that one and only one

operation is executed on a machine per each time step. Equation (A.3) allows each machine to execute only one operation at a time. Equation (A.4) sets the objective function as larger than operations completion times. Equation (A.5) takes care of the precedence constraint among operations of the same job.

2.2 Disjunctive Model

The decision variables in this model are as follows:

- s_{mj} is an integer variable and models the start time of job j on machine m.

$$\text{minimize } T_{max} \text{ subject to}$$

$$s_{mj} \geq 0 \qquad \forall j \in J, m \in M \tag{A.6}$$

$$T_{max} \geq s_{o_k^j,j} + d_{o_k^j,j} \qquad \forall j \in J \tag{A.7}$$

$$s_{o_i^j,j} \geq s_{o_{i-1}^j,j} + d_{o_{i-1}^j,j} \qquad \forall j \in J, i = 2, \ldots, k \tag{A.8}$$

$$s_{mj} \geq s_{mj'} + d_{mj'} \vee s_{mj'} \geq s_{mj} + d_{mj} \qquad j, j' \in J, j \leq j', m \in M \tag{A.9}$$

In this model equation (A.6) restricts variables domain to be larger than or equal to zero, as they represent the start time of operations and thus they can not be negative. Equation (A.7) impose the objective function to be lager than or equal to the start time of the last operation of each job plus its duration. Equation (A.8) is about precedence constraints among the operations belonging to the same job: one can not start until the previous one is over. Equation (A.9) takes care of resource sharing by stating that two operations sharing the same resource cannot take place at the same time: either one starts once the other is over or the other way around.

2.3 Rank-Based Model

In this model the focus is on the machine side: each machine has as many positions as operations in a job or, in other words, a position is the cardinal step in the execution sequence of all jobs. Finding a schedule for this model

means to find in which position a job is executed on a certain machine. The decision variables are defined as follows:

- x_{mjt} is a Boolean variable indicating whether job j is executed on machine m at the t-th position
- s_{mt} is an Integer variable representing the starting time of position t of machine m

minimize T_{max} subject to

$$s_{mt} \geq 0 \qquad \forall m \in M, t = 1 \dots k \tag{A.10}$$

$$x_{o_k^j jt} \rightarrow T_{max} \geq s_{o_k^j t} + d_{o_k^j j} \qquad \forall j \in J, t = 1 \dots k \tag{A.11}$$

$$x_{mjt} \rightarrow s_{mt} + d_{mj} \leq s_{m,t+1} \qquad \forall t = 1 \dots k, m \in M, j \in J \tag{A.12}$$

$$(x_{o_{i-1}^j jt_1} \wedge x_{o_i^j jt_2}) \rightarrow s_{o_{i-1}^j t_1} + d_{o_{i-1}^j j} \leq s_{o_i^j t_2} \qquad \forall t_1, t_2 = 1 \dots k, i = 2 \dots k, j \in J \tag{A.13}$$

$$\left(x_{mjt} \rightarrow \bigwedge_{t'=0}^{t-1} \neg x_{mjt'} \wedge \bigwedge_{t'=t+1}^{k} \neg x_{mjt'}\right) \wedge \bigvee_{k=0}^{k} x_{mjt} \qquad \forall t = 1 \dots k, m \in M, j \in J \tag{A.14}$$

$$\left(x_{mjt} \rightarrow \bigwedge_{j'=0}^{j-1} \neg x_{mj't} \wedge \bigwedge_{j'=j+1}^{n} \neg x_{mj't}\right) \wedge \bigvee_{t=0}^{k} x_{mj't} \qquad \forall t = 1 \dots k, m \in M, j \in J \tag{A.15}$$

Equation (A.10) restricts the start variable domain to the natural numbers. Equation (A.11) sets the objective function. Equation (A.12) ensures the precedence constraint among the operations executed on a machine, while equation (A.13) takes care of the operations precedence within the same job. Equation (A.14) ensures that each job can be assigned only once to a certain machine, while equation (A.15) states that a position can be assigned only to one job.

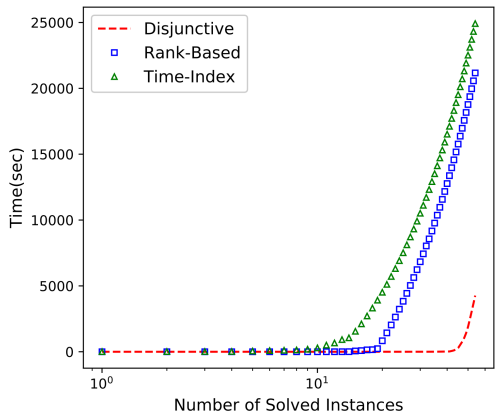

Figure 1: Performance comparison between Disjunctive, Time-Index and Rank-Based models over the benchmark instances. The maximum time allowed for each instance is 600 seconds.

3 Experiments

The solvers whose performance were compared are Z3-4.6.0, Gurobi-7.5.2 and OptiMathSAT-1.5.0. The time limit is 600 seconds. Solvers are run in their default setting. The instances used for the comparison are either generated through an instance generator or taken from benchmark sets. All the experiments were performed on an *Intel Core i7 6700K, 4.0 GHZ, 32GB RAM* running *Ubuntu-16.04*.

An instance is a matrix of integers where each row represents a job; for each row the odd elements represent the machine needed to execute the operation whose duration is pointed out by the next even element. The execution order is given by the position within the row. The instances used in the experimental phase are:

- Generated instances: instances generated according to Taillard instance generator[21] of small-medium size (from 3x3 to 9x9);
- Lawrence [22]: forty instances of increasing size from 10x5 to 30x10;
- Applegate and Cook [23]: ten instances of size 10x10;

Table 1: Comparison of models implemented using Z3. The time showed in the table is the geometric mean calculated over all the instances belonging to the category they refer to. For each class the number of solved instances (out of the total number of instances belonging to such class) is given. The symbol '-' means that no instance has been solved. The symbol '*' means that the time limit for the model translation into SMT-lib2 has been exceeded.

	Disjunctive		Rank-Based		Time-Index	
Problems	**Time**	**Opt**	**Time**	**Opt**	**Time**	**Opt**
Generated Instances						
3x3	0.01	5/5	0.18	5/5	3.66	5/5
4x4	0.01	5/5	0.110	5/5	32.76	5/5
5x5	0.02	5/5	1.196	5/5	164.67	5/5
6x6	0.04	5/5	38.904	5/5	528.31	2/5
7x7	0.12	5/5	535.25	1/5	-	0/5
8x8	0.27	5/5	-	0/5	*	*
9x9	0.18	5/5	-	0/5	*	*
Applegate Instances						
10x10	7.28	10/10	-	0/5	*	*
Taillard Instances						
15x15	240.84	7/10	-	0/5	*	*

- Storer [24]: five instances of size 20x10;
- Taillard [21]: ten instances of size 15x15 and ten of size 20x15.
- Fisher&Thompson [25]: one instance of size 20x5.

3.1 Models Comparison

In this phase the three models presented in the previous section are compared Z3 (Figure 1). We decided to employ the geometric mean to reduce the effect of outliers. The instance generation has been carried out by randomly assigning values belonging to the interval $[1, 20]$ to the operations. Five instances have been generated per each class going from the size 3x3 to 9x9.

The models have been created through the Python API for Z3 and then translated into the SMT-lib2 format [26] to be run directly and avoid possible delays due to the conversion while measuring the execution time. Since the Time-Index model scales up very bad in size an additional timeout has been

set for the model translation into SMT format and the * symbol in the table denotes that such time limit has been exceeded. The results show that the Time-Index model can easily solve very small-size instances but it scales bad and provides no solution for problems larger than 6x6. The Rank-Based model is more efficient in terms of time but it can solve only some instances more than Time-Index.

The Disjunctive model, on the other hand, takes only few milliseconds to solve the smaller instances and it stays far below one second while dealing with instances up to 9x9 size. The average time increases by over forty times for solving Applegate instances due to some outliers that take over one minute but the result is still remarkable if compared to the other two models, even when it comes to Taillard Instances, although only seven out ten instances can be solved within ten minutes.

An additional test has been run on some hard-to-solve instances, to check how the best solution increases over time, as shown in Figure 2. Usually a solution close to the optimal is found quickly but then it is hard to improve it. This might be due to the solver getting stuck in some local optimum.

3.2 Solvers Comparison

The second phase is about comparing different solvers using the Disjunctive model. When running the Disjunctive model on Applegate instances, OptiMathSAT could solve six out of ten of them and it was on average four times slower than Gurobi, which could solve seven. Z3 could solve all of them in less than eight seconds each. All the solvers could easily deal with the 10x5 instances from Lawrence and while Z3 solution was almost instantaneous Gurobi and OptiMathSAT had similar performance taking around twenty seconds to produce a solution. The only other class of instances Gurobi and OptiMathSAT were able to find a solution for is Lawrence 10x10 and also in this case Gurobi was quite faster than OptiMathSAT (around ten times) and both much slower than Z3, which was also able to solve some of Lawrence instances 15x10 in less than five minutes, 15x15 in less than three, and seven of the Taillard instances 15x15, as shown also in the previous section. No other solution was found within the given time.

Table 2: Comparison of SMT and MILP solvers running the Disjunctive Model over the benchmark instances. The time showed in the table is the geometric mean calculated over all the instances belonging to the category they refer to. For each class the number of solved instances (out of the total number of instances belonging to such class) is given. The symbol '-' means that no instance has been solved.

	Z3		OptiMathSAT		Gurobi	
Problems	**Time**	**Opt**	**Time**	**Opt**	**Time**	**Opt**
Applegate Instances						
10*10	7.28	10/10	276.14	6/10	60.13	7/10
Lawrence Instances						
10x5	0.43	5/5	18.41	5/5	16.21	5/5
15x5	-	0/5	-	0/5	-	0/5
20x5	-	0/5	-	0/5	-	0/5
10x10	0.56	5/5	215.85	5/5	18.20	5/5
15x10	263.61	3/5	-	0/5	-	0/5
20x10	-	0/5	-	0/5	-	0/5
30x10	-	0/5	-	0/5	-	0/5
15x15	156.37	2/5	-	0/5	-	0/5
Storer Instances						
20x10	-	0/5	-	0/5	-	0/5
Taillard Instances						
15x15	240.84	7/10	-	0/10	-	0/10
20x15	-	0/10	-	0/10	-	0/10

3.3 Results Discussion

The results presented in the previous section show a big difference in performance between the two SMT solvers. When compared to Gurobi, Z3 is typically faster and can provide an optimal solution to larger instances within the given time limit. OptiMathSAT, although employing the same technology, is considerably slower than Z3 and, in most cases, also than Gurobi. Since Z3 and OptimathSAT have shown comparable performance in previous works [19], it might be the case that Z3 heuristic and linear optimization algorithms suits better the JSP problem than OptiMathSAT's. By having a look under the hood of νZ we found out it employs a portfolio of different approaches to solve optimization problems [18]. Among them are some very efficient algorithms to deal with linear arithmetic using Simplex over non-standard numbers to find unbounded objectives, as explained by Z3 developers in [27].

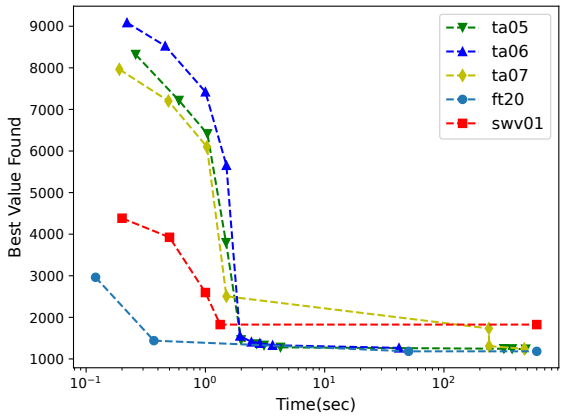

Figure 2: Relation between time and best value found for benchmark instances 'ta05, 'ta06', 'ta07', 'ft20', 'swv01' running the Disjunctive model in Z3 during 3600 seconds.

This method allows to find the solution in one call without need for iterating over potentially many of them.

Unlike νZ, that uses Z3 has a black box and it is built on top of it, OptiMathSAT has an inline architecture that calls the SMT solver only once and within it the SAT solver is then modified to handle the optimization. An insight about the optimizing algorithms running within the solver is given in Sebstiani's work [28]. Improved versions of the Branch&Bound algorithm are developed to exploit the features of MathSAT5 when dealing with linear algebra over Reals (LRA), integers (LIA) or a combination of both ($LRIA$).

4 Conclusions

In this paper we have compared three models for JSP suitable for optimizing SMT-solvers. We also compared the disjunctive model formulation in Z3 and OptiMathSAT with Gurobi, a state-of-the-art commercial MILP solver. On the benchmark examples Z3 outperforms Gurobi, and Gurobi outperforms OptiMathSAT. The results are very interesting because SMT-solvers are general purpose solvers that can easily

include additional constraints that are relevant in industrial applications

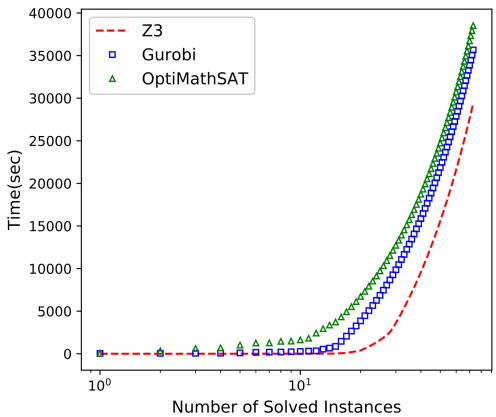

Figure 3: Performance comparison between SMT solvers Z3 and OptiMathSAT and MILP solver Gurobi over the benchmarck instances. The maximum time allowed for each is 600 seconds.

making them an attractive choice for real applications. There exist options such as dedicated algorithms based on CP and local search that provide better performance; nevertheless the results shown in this paper classify SMT-solvers, and Z3 in particular as a good alternative, especially since it is available under an open-source license (MIT License) and hence it is possible to use it for commercial purpose. Today, the licensing costs for commercial MILP-solvers can be substantial, something that will restrict the numbers of applications where scheduling can be motivated. Since optimization was added very recently to SMT-solvers and the research on SMT-solvers is a very active field with rapid progress it is reasonable to expect that the performance of optimizing SMT-solvers will continue to improve.

References

[1] A. S. Jain and S. Meeran, "Deterministic job-shop scheduling: Past, present and future," *European Journal of Operational Research*, vol. 113, no. 2, pp. 390–434, 1999.

[2] Y. Nagata and I. Ono, "A guided local search with iterative ejections of bottleneck operations for the job shop scheduling problem," *Computers & Operations Research*, vol. 90, pp. 60–71, 2018.

[3] J. C. Beck, T. Feng, and J.-P. Watson, "Combining constraint programming and local search for job-shop scheduling," *INFORMS Journal on Computing*, vol. 23, no. 1, pp. 1–14, 2011.

[4] W.-Y. Ku and J. C. Beck, "Mixed integer programming models for job shop scheduling: A computational analysis," *Computers & Operations Research*, vol. 73, pp. 165–173, 2016.

[5] R. Lima, "IBM ILOG CPLEX–What is inside of the box," in *Proc. 2010 EWO Seminar*, 2010, pp. 1–72.

[6] Gurobi. "Gurobi optimizer reference manual." (2015), [Online]. Available: `http://www.gurobi.com`.

[7] A. Gleixner, L. Eifler, T. Gally, *et al.*, "The SCIP optimization suite 5.0," eng, ZIB, Takustr.7, 14195 Berlin, Tech. Rep. 17-61, 2017.

[8] "Miplib 2010: Mixed integer programming library version 5," English (US), *Mathematical Programming Computation*, vol. 3, no. 2, pp. 103–163, 2011, ISSN: 1867-2949.

[9] J. F. Gonçalves, J. J. de Magalhães Mendes, and M. G. Resende, "A hybrid genetic algorithm for the job shop scheduling problem," *European Journal of Operational Research*, vol. 167, no. 1, pp. 77–95, 2005.

[10] E. Balas and A. Vazacopoulos, "Guided local search with shifting bottleneck for job shop scheduling," *Management science*, vol. 44, no. 2, pp. 262–275, 1998.

[11] A. S. Jain and S. Meeran, "A state-of-the-art review of job-shop scheduling techniques," Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, Tech. Rep., 1998.

[12] J. Franco and J. Martin, "A history of satisfiability.," *Handbook of satisfiability*, vol. 185, pp. 3–74, 2009.

[13] C. Barrett and C. Tinelli, "Satisfiability modulo theories," *Handbook of model checking*, pp. 305–343, 2018.

[14] L. De Moura and N. Bjørner, "Satisfiability modulo theories: Introduction and applications," *Commun. ACM*, vol. 54, no. 9, pp. 69–77, Sep. 2011, ISSN: 0001-0782.

[15] ——, "Z3: An efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2008, pp. 337–340.

[16] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani, "The MathSAT5 SMT solver.," in *TACAS*, N. Piterman and S. A. Smolka, Eds., ser. Lecture Notes in Computer Science, vol. 7795, Springer, 2013, pp. 93–107, ISBN: 978-3-642-36741-0.

[17] C. Barrett, C. L. Conway, M. Deters, *et al.*, "CVC4," in *Proceedings of the 23rd International Conference on Computer Aided Verification*, ser. CAV'11, Snowbird, UT: Springer-Verlag, 2011, pp. 171–177, ISBN: 978-3-642-22109-5.

[18] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νZ-an optimizing SMT solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2015, pp. 194–199.

[19] R. Sebastiani and P. Trentin, "OptiMathSAT: A tool for optimization modulo theories," in *International Conference on Computer Aided Verification*, Springer, 2015, pp. 447–454.

[20] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, May 1976, ISSN: 0364-765X.

[21] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.

[22] S. Lawrence, "Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement)," *Graduate School of Industrial Administration, Carnegie-Mellon University*, 1984.

[23] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[24] R. H. Storer, S. D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Management science*, vol. 38, no. 10, pp. 1495–1509, 1992.

[25] H. Fisher, "Probabilistic learning combinations of local job-shop scheduling rules," *Industrial scheduling*, pp. 225–251, 1963.

[26] C. Barrett, A. Stump, C. Tinelli, *et al.*, "The SMT-lib standard: Version 2.0," in *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England)*, vol. 13, 2010, p. 14.

[27] N. Bjørner and A.-D. Phan, "νZ-maximal satisfaction with Z3.," *SCSS*, vol. 30, pp. 1–9, 2014.

[28] R. Sebastiani and P. Trentin, "Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions," in *Proc. Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'15*, ser. LNCS, vol. 9035, Springer, 2015.

PAPER B

On the Use of Equivalence Classes for Optimal and Sub-Optimal Bin Packing and Bin Covering

Sabino Francesco Roselli, Fredrik Hagebring, Sarmad Riazi, Martin Fabian, Knut Åkesson

2021 IEEE Transactions on Automation Science and Engineering (TASE)
vol. 18, no. 1, pp. 369-381, Jan. 2021.
 DOI: 10.1109/TASE.2020.3022986

The layout has been revised.

Abstract

Bin packing and *bin covering* are important optimization problems in many industrial fields, such as packaging, recycling, and food processing. The problem concerns a set of *items*, each with its own *value*, that are to be sorted into *bins* in such a way that the total value of each bin, as measured by the sum of its item values, is not above (for packing) or below (for covering) a given *target* value. The optimization problem concerns minimizing, for bin packing, or maximizing, for bin covering, the number of bins. This is a combinatorial NP-hard problem, for which true optimal solutions can only be calculated in specific cases, such as when restricted to a small number of items. To get around this problem, many sub-optimal approaches exist. This paper describes formulations of the bin packing and covering problems that allows to find the true optimum for instances counting hundreds of items using general purpose MILP-solvers. Also presented are sub-optimal solutions that come within less than 10% of the optimum, while taking significantly less time to calculate, even ten to 100 times faster, depending on the required accuracy.

Note to Practitioners

A typical case for bin covering is in food processing where food items are automatically sorted into trays of similar weight, so that the overweight is minimized. Another application is in recycling, where items like batteries should be put in crates of similar weight, so that the crates do not exceed a target weight, due to later manual handling, but at the same time we want as few crates as possible. This is a bin packing problem. On an industrial scale these tasks are fully automated. Though modern software tool's efficiency to solve bin sorting problems have increased significantly in later years, the problems are inherently tough in the sense that the solution time grows exponentially with the number of items. This limits the problem sizes that can be solved to optimality within reasonable time. Therefore, much research has focused on heuristic rules that give reasonable solving times while not giving

the true optimal number of bins. However, in many cases the true optimal solution is preferable, and sometimes even necessary, so this is an industrially interesting problem. This paper describes an approach to solve the bin packing and covering problems to the true optimum that increases the limit of the number of items that can typically be handled. This is done by observing that items of same value need not be distinguished. Instead, we can formulate packing/covering problems over item values rather than individual items, and sort integer numbers of these values into bins, which allows to solve to optimum for more than 500 hundred items in reasonable time. In addition, by redefining what we mean by *same value*, we can consider more items to have the same value and achieve even better computational efficiency.

1 Introduction

The (one-dimensional) *bin sorting* problem concerns sorting items with given values into *bins* such that the value of a bin, counted as the sum of its included values, conforms to a specified *target* value, while at the same time optimizing the total number of bins. Two (dual) variants of this problem exist, *bin packing* [1] where the bin values cannot go over, and *bin covering* [2] where the bin values cannot go under, respectively, the target value.

The problems are NP-hard [1] combinatorial optimization problems, meaning that there is no general algorithm that can solve either problem for an arbitrary number of items in reasonable time.

Due to this, different heuristic algorithms to provide sub-optimal solutions while guaranteeing a bound from the optimum and having polynomial complexity have been a long-time active field of research. Recent surveys of related problems are presented in [1] for approximative algorithms and in [3] for exact algorithms.

Recent work on bin sorting is based on branch-and-price based algorithms, see e.g. [4]. Pseudo-polynomial formulations, [5]–[7], allow a more compact formulation and avoid the complexity introduced in the implementation of branch-and-price based algorithms. Of interest in some practical applications is also temporal extensions, [8], where the capacity of a bin must be consumed within a given time window.

In our previous work on the subject [9], we presented a model to solve the bin covering problem to optimality by means of *Mixed Integer Linear*

Programming (MILP), and we showed through a computational analysis of more than 800 problem instances that a MILP solver is able to handle quickly large sized instances. In this work we extend the analysis to also bin packing, and we provide mathematical proof of the validity of our claims on top of which we implemented the above-mentioned formulation. We also present a sub-optimal approach based on a simplification of the original instances that exploits the feature of our new formulation and guarantees close to optimal results.

In industrial problems there are often additional constraints that are not included in text book formulations of the sorting problems. An example from an industrial application: for bin covering problems it might be allowed to go a few percent below the target weight for certain bins as long as the average bin value is on or above target. Algorithms that are tailor made for textbook formulations of the sorting problems might have problems to generalize to such modified versions of the problem. In industrial applications general purpose solvers are thus often preferred due to their ability generalize to new problem formulations. So, though we in this paper treat only the text book versions of bin sorting, we do so by focusing on formulations that can be given as input to general-purpose MILP solvers, and we evaluate the efficiency of these formulations.

First is presented the standard formulation that can be found in most text books. This formulation was first introduced by [10] and is typically useful only for a small number of items. Then is given the "subset" formulation, which removes the identities of the bins and thus allows to solve for a much larger number of items and bins. Thirdly is given the "equivalence class" formulation, which further removes the identities between items of same values, and hence allows to solve for even larger numbers of items and bins. As explained later on in the paper, the idea of *equivalence classes* leads to define combinations of items that meet the target requirement (i.e. the cumulative value of such items is below the target for the bin packing and above it for the bin covering); such combinations are given the name of *packages* in this paper and, to the best of our knowledge, they have first been introduced by [11] to implement a specific purpose algorithm to solve bin covering problem, and then used in [12] to implement a branch and price algorithm, while we use it to develop a linear-integer model for a general purpose solver. Also, we improve the concept by defining a subset of packages among which bins can

be selected that still leads to the optimal solution. In other words, we do not need to enumerate all possible combinations of items in order to find the optimal solution, but only a rather small portion of it.

The contributions to this paper are: (i) improving the concepts of equivalence classes by introducing the notion of *skinny* and *fit* bins for the bin covering and the bin packing respectively; proving that the the optimal solution of an instance of the bin packing (covering) is only composed by *fit* (*skinny*) bins; (iii) develop a heuristic method for both the bin covering and bin packing problem, based on equivalence classes, that significantly reduces the computation time while still guaranteeing close to optimum results; (iiii) evaluating the method against other algorithms for bin sorting problems by running it over different sets of benchmark instances.

In the next section the general bin sorting problem is described, giving three different MILP formulations, two of which exploit the fact that prospective bins can be pre-calculated to make the MILP solver's job easier. Then Section 3 presents how the combinatorial explosion can be further mitigated by restricting the number of pre-calculated bins, while still guaranteeing optimal solutions. Section 4 then describes how the number of pre-calculated bins can be made even smaller, but then not guaranteeing optimal solutions. The experimental results of Section 5 shows the computational benefits of both the optimal formulations, and the sub-optimal formulation that comes within less than 10% of the optimum, while achieving a significant reduction in computation time. The paper is concluded in Section 6.

2 Bin Sorting

Bin sorting is a generic term for the two (dual) problems of bin *packing* and bin *covering*. The bin sorting problem concerns a set of *items* $V = \{v_1, v_2, \ldots, v_n\}$, each with a *value* so that there can be defined an ordering between the items, such that $v_1 \geq v_2 \geq \cdots \geq v_n$. For notational simplicity, except for in a few places, the distinction between an item and its value will not be made; note though that items are unique, whereas two items can have the same values. Given a subset $V' \subseteq V$ we denote its minimum and maximum values as $\min(V')$ and $\max(V')$, respectively.

A *bin* $b_j \subseteq V$ is a subset of V. For a bin b_j we can define its value B_j as the sum of the item values it contains, that is, $B_j = \sum_{v_i \in b_j} v_i$.

The bin sorting problem can now be defined as a tuple $\langle V, t, \bowtie \rangle$, where t is a target value that defines a bound on the bin values, and $\bowtie$ is $\leq$ for bin packing, and $\geq$ for bin covering. The problem is now to find an optimal solution $B_{opt} = \{b_1, b_2, \ldots, b_m\}$, which is a partition of V with the minimum, for packing, or maximum, for covering, number m of bins, such that $\forall b_j \in B_{opt},\ B_j \bowtie t$. It is assumed that $\sum_{v_i \in V} v_i > t$, and $\forall v_i \in V\ v_i < t$.

Since we in large parts of the paper simultaneously deal with both covering and packing, we have introduced a non-standard notation of our own (such as "target" t instead of "capacity" c). This so, since the communities dealing with the respective problems do not always agree on the notation. Furthermore, the term "bin covering" is sometimes used to denote a different problem, where the number of bins is fixed and the problem is maximizing the number of packed items while not exceeding the target value for any bin [13].

2.1 The Standard Formulation

One way to formulate the bin sorting problem is as a *mixed linear integer programming* (MILP) problem, where the decision variables represent bins and the allocation of items to the bins. Let b_j $(j = 1, \ldots, n)$ be 0-1-variables representing whether a certain bin is used $(b_j = 1)$ or not $(b_j = 0)$, and let x_{ij} $(i, j = 1, \ldots, n)$ be 0-1-variables representing whether the value v_i is assigned to the j'th bin $(x_{ij} = 1)$, or not $(x_{ij} = 0)$. The MILP problem can then be formulated as:

$$\min / \max \sum_{j=1}^{n} b_j \text{ subject to} \tag{B.1}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = 1, \ldots, n \tag{B.2}$$

$$\sum_{i=1}^{n} x_{ij} \cdot v_i \bowtie b_j \cdot t \qquad \forall j = 1, \ldots, n \tag{B.3}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j = 1, \ldots, n \tag{B.4}$$

$$b_j \in \{0, 1\} \qquad \forall j = 1, \ldots, n \tag{B.5}$$

The objective function (B.1) is the sum over all the variables representing whether a certain bin is used or not, and since these are binary 0-1-variables,

the sum is the number of used bins; this sum is to be minimized for bin packing and maximized for bin covering. Constraint (B.2) guarantees that each item is assigned to exactly one bin. Constraint (B.3) guarantees that the value of each used bin is on or below ($\bowtie$ is $\leq$) the target value t for bin packing, and on or above ($\bowtie$ is $\geq$) the t for bin covering. Constraints (B.4) and (B.5) define the domains of the decision variables.

For bin covering, Constraint (B.2) can actually be relaxed to ≤ 1, since not all values are necessarily placed in some bin. However, such *surplus* values (see below) cannot constitute a bin on their own, and since there is no upper bound on the bins, the surplus values may be put on any bin without altering the optimal solution. In fact, with a rigorous definition of the surplus values, we can always remove the surplus values from the optimal solution, and the surplus-free solution will still be optimal.

2.2 The Subset Formulation

A less trivial approach to formulate the bin sorting problem as a MILP problem is to enumerate the prospective bins by sort the items into *packages* that fulfill the target constraint, and then formulate a problem of choosing the smallest or largest number of such packages. If we generate *all possible* such packages, the MILP-solver will have freedom enough to find the optimal number of packages.

Let $p_j \subseteq V$ be a package such that $\sum_{v_i \in p_j} v_i \bowtie t$. Note that contrary to bins, different packages may share items, that is, for some packages p_i and p_j with $i \neq j$ it may hold that $p_i \cap p_j \neq \varnothing$. Let $P_i = \{p_j | v_i \in p_j\}$ be the set of all packages that include the item v_i. We call the elements of P_i *overlapping*, and there are n such sets.

Given the set of k generated packages, and with a slight abuse of notation we use p_j to denote a 0-1-variable representing whether the package p_j is used ($p_j = 1$) or not ($p_j = 0$), we can formulate the MILP problem as:

$$\min / \max \sum_{j=1}^{k} p_j \text{ subject to} \tag{B.6}$$

$$\sum_{p_j \in P_i} p_j = 1 \qquad \forall i = 1, \ldots, n \tag{B.7}$$

$$p_j \in \{0, 1\} \qquad \forall j = 1, \ldots, k \tag{B.8}$$

Similarly to (B.1), the objective function (B.6) sums over all the variables representing whether a package is used or not, and since these are 0-1-variables, the sum is the number of used packages; this sum is to be minimized, or maximized, for bin packing and bin covering, respectively. Constraint (B.7) guarantees that exactly one of the overlapping packages is used, which prohibits multiple inclusion of the same item into the optimal solution, and so guarantees that the chosen set of packages partition V (this is what makes the chosen packages bins). Again for the bin covering the constraint may be relaxed into a less then equality, since the maximisation will make sure that as many packages as possible are chosen; on the other hand, without the more restrictive constraint, the bin packing problem would always yield a solution counting zero bins. Constraint (B.8) simply defines the domains of the p_j variables.

2.3 Equivalence class formulation

Though the subset formulation of Section 2.2 goes a long way to mitigate the computational complexity, observing that for large bin sorting problems we can have, and typically do have, many items with equal values, we can give an even more compact problem formulation, where equal values are not viewed as distinct items, but rather as a single item with a multiplicity equal to the number of actual such same-valued items. This is done by collecting equal items into *equivalence classes*, and instead of enumerating each item of such a class, formulate the optimization problem over integer decision variables related to the number of items in each equivalence class.

Consider a bin sorting problem $\langle V, t, \bowtie \rangle$. Let an equivalence class E_q be a subset of equal valued items of V; that is, $E_q = \{v_i \in V | v_i = w\}$ for a fixed value w. Obviously, $\min(E_q) = w$. Let p denote the number of all equivalence

classes. The set of p equivalence classes partition V.

We call a tuple $\langle E_q, f_q \rangle$ of an equivalence class E_q and a *factor* f_q, a *selection*. The factor is used to denote the number of items from the equivalence class that are selected in a certain situation. Of course, $0 \leq f_q \leq |E_q|$, and obviously there is a finite number of distinct selections.

Let a *package class* $PC_i = \{\langle E_1, f_{1,i} \rangle, \ldots, \langle E_p, f_{p,i} \rangle\}$ be a set of selections over all equivalence classes, such that

$$\sum_{\langle E_q, f_{q,i} \rangle \in PC_i} \min(E_q) \cdot f_{q,i} \bowtie t, \tag{B.9}$$

where $\bowtie$ is $\leq$ for bin packing and $\geq$ for bin covering, and the objective is to minimize and maximize, for packing and covering, respectively. Let k denote the number of all possible package classes.

Since all package classes contain all equivalence classes, albeit many with a zero factor, all package classes overlap, which then becomes an uninteresting observation (contrary to Section 2.2).

Given a set of k generated package classes, and with some abuse of notation let PC_i be an integer that represents how many "instances" of the package class PC_i that are included in the optimal solution, then the optimization problem can be formulated as:

$$\min / \max \sum_{i=1}^{k} PC_i \text{ subject to} \tag{B.10}$$

$$\sum_{i=1}^{k} f_{q,i} \cdot PC_i = |E_q| \qquad \forall q = 1, \ldots, p \tag{B.11}$$

$$PC_i \geq 0 \qquad \forall i = 1, \ldots, k \tag{B.12}$$

The objective function (B.10) sums over all the variables representing the number of each package class instance; this sum is to be minimized for bin packing and maximized for bin covering. Constraint (B.11) sums for each equivalence class over all the package classes and multiplies with the factor for each respective package class, to get the total number of used values from the specific equivalence class. Naturally, this number cannot exceed the number of values in the equivalence class for the package class, and has to be exactly equal to the number of values in the equivalence class in order to avoid trivial

solutions with zero bins. Constraint (B.12) simply sets zero as the lower bound for the number of instantiations of the respective package classes.

An upper bound for PC_i could be pre-calculated, but it is not clear whether this will have any impact on the computational complexity, and this has not been investigated.

For brevity, we will in the following use the term *package* in place of *package class*.

3 Optimal Solutions

Though theoretically possible, generating *all* packages is practically intractable; we can do this only for small n. And though generating *all* possible package classes is more tractable than generating all packages, it still amounts to a huge computational effort for large bin sorting problems. However, we can calculate the packages in a more clever way, by observing that the optimization criterion really means that the resulting bins of an optimal solution should be as close to the target value as possible. Calculating such packages saves a lot of computational effort, as is shown in Section 5.

In this section we will argue for why and how we can calculate a specific subset of all possible packages, but still get an optimal solution from the subset and equivalence class formulations of Section 2. For clarity, we will here treat packing and covering separately in their own subsections.

In both cases, we have a bin sorting problem $\langle V, t, \leq \rangle$ for packing, and $\langle V, t, \geq \rangle$ for covering. The total value over all n items of V is $W = v_1 + v_2 + \cdots + v_n$. A *feasible* solution $f_t = \{b_1, b_2, \ldots, b_m\}$ to a bin sorting problem, is a solution where for each bin, $B_j \leq t$ for packing, and $B_j \geq t$ for covering, and an *optimal* solution is a feasible solution where the number of bins m is minimized for packing, and maximized for covering.

Let us also note the distinction between *bins* and *packages*. Bins partition V so that no item in V can be in more than one bin, whereas packages can share items; it is the task of the bin sorting solver to select among the packages so that a single item does not appear in more than one bin.

3.1 Bin Packing

A feasible solution $f_t = \{b_1, b_2, \ldots, b_m\}$ for a bin packing problem $\langle V, t, \leq \rangle$ is said to be *true* if all items of V are sorted.

For a feasible solution, all bins b_j are on or below target, that is $B_j \leq t$. For bin packing we want to minimize the number of bins. Thus, it seems to make sense to have bins that are as close to the target (but not above) as possible.

Definition 1: *A bin (or package) is said to be* fit *if adding the least valued item from* V *gets it above the target value. That is, a bin b_j is fit if*

$$B_j + \min(V) > t.$$

Definition 2: *Let V^v be a set of* virtual *items, such that $V^v \cap V = \varnothing$ and $v_k = \min(V)$ for all $v_k \in V^v$.*

The virtual items are not in the original problem formulation but are introduced as possible items that can be put in bins to complete a bin into a fit bin, in order to guarantee fit solutions. However, we show later that virtual items can be removed from a solution and thus a *true* solution can be generated.

A feasible solution $f_f = \{b_1, b_2, \ldots, b_m\}$ for a bin packing problem $\langle V, t, \leq \rangle$ is said to be *fit* if all items of V, plus an arbitrary number of virtual items, are sorted, and all bins in f_f are fit.

Lemma 1: *For a bin packing problem $\langle V, t, \leq \rangle$, a feasible* true *solution f_t exists if and only if a feasible fit solution f_f exists.*

Proof. First we show that to a feasible true solution f_t we can add virtual items to non-fit bins to make a feasible fit solution f_f.

Assume a non-fit bin b_j, thus $B_j + \min(V) \leq t$. Add $\lfloor (t - (B_j + \min(V)))/\min(V) \rfloor$ number of virtual items to b_j, the bin is now fit by definition since adding one additional virtual item makes the value of the bin be larger than the target value t. This can be done for any non-fit bin in f_t.

Second, we show that we can remove (virtual) items from the bins of f_f and get feasible *true* solution.

Assume a fit solution f_f. Let B^f be the sum of the values of all bins for a feasible fit solution f_f. Then the total number of virtual items is equal to $\lfloor (B^f - W)/\min(V) \rfloor$. If this number of virtual items are removed from the bins in f_f, the total number of items of value $\min(V)$ in all bins will be equal to $|E_n|$ and thus the modified solution will be *true*. Note that removing virtual items from a bin can only decrease its value, and since a fit bin is by definition

already below the target, so will the *reduced* bin be. The total number of items of value $\min(V)$ will be equal to or larger than $\lfloor {}^{(B^f-W)}/_{\min(V)} \rfloor$, thus it is possible to remove $\lfloor {}^{(B^f-W)}/_{\min(V)} \rfloor$ items of value $\min(V)$. Consider any set of bins that is the result of removing $\lfloor {}^{(B^f-W)}/_{\min(V)} \rfloor$ items of value $\min(V)$ from the bins of the fit solution f_f. The set of reduced bins result in a feasible solution for the *true* problem, since all bins have a value less than the target value and the number of items for every value will be equal to the number of values in the equivalence class for the same value. □

Theorem 1: *Let B^t_{opt} be an optimal true solution to the bin packing problem $\langle V, t, \leq \rangle$, and let B^f_{opt} be an optimal fit solution to the same problem. Then*

$$|B^t_{opt}| = |B^f_{opt}|.$$

Proof. If there exists a feasible solution for the fit problem that is optimal, then no better solution than that exists and, according to Lemma 1, there must exist a feasible solution for the *true* problem that yields the same result and thus no better solution can exist. □

3.2 Fit package generation

As mentioned, a major issue with the equivalence classes approach to solve bin packing problems is the computation of all package classes that might form part of the optimal solution. Computing and then filtering the power set of V is computationally heavy even for relatively small size problems, therefore a less demanding procedure is required. Given the aforementioned notions, the computation of all *fit* package classes can be formulated as a Constraint Satisfaction Problem (CSP).

Regard the bin packing problem $\langle V, t, \leq \rangle$, with the equivalence classes E_q $(q = 1, \ldots, p)$. Let f_q $(q = 1, \ldots, p)$ be the factor for E_q, that is, an integer variable representing how many values from E_q that are chosen to form a package class. Let F be an integer number such that $F = \lfloor {}^{t}/_{\min(V)} \rfloor$. The

CSP formulation is as follows:

$$0 \leq f_q \leq F \qquad \forall q = 1, \ldots, p \tag{B.13}$$

$$\sum_{\substack{j=1 \\ q \neq j}}^{p} \Big(\min(E_j) \cdot f_j + (f_q + 1) \cdot \min(E_q) \Big) > t$$

$$\forall q = 1, \ldots, p \tag{B.14}$$

Constraint (B.13) limits the factor for each equivalence class to be at most as large as the value F; constraint (B.14) states that the sum of values contained in a *fit package* goes above the target value as soon as we increase the value of any factor by *one*.

Finding a satisfiable solution to this CSP problem means to find a combination of values that, together will result in a *fit* package class. To obtain the whole set of fit package classes, it is possible to set up another problem with the same constraints, plus one constraint ruling out the solution just found. Let $S = \{f_1^*, \ldots, f_p^*\}$ be the solution of the CSP problem, where $f_i^* \ \forall i = 1, \cdots, p$ is the factor selected for the equivalence class p, then the additional constraint is:

$$\neg\Big(\bigwedge_{f_i^* \in S} (f_i = f_i^*) \Big) \tag{B.15}$$

Constraint B.15 ensures that the solution found in the previous iteration cannot be selected in the current one, so that the solver has to find a new one. In order to find the complete set of *fit* package classes, one has to set up a loop and, for each iteration, add the constraint to rule out the last solution found for the new CSP problem. The loop goes on until the problem becomes unsatisfable, which means that all package classes have been found.

Example of Equivalence Class Formulation Consider the bin packing problem $\langle V, t, \leq \rangle$, with $V = \langle 50, 50, 40, 40, 10, 10 \rangle$ and $t = 100$.

There are three equivalence classes, all of size 2:

$E_1 = \langle 50, 50 \rangle$

$E_2 = \langle 40, 40 \rangle$

$E_3 = \langle 10, 10 \rangle$

And there will also be 8 virtual items of value 10. We need to be able to form feasible packages by using the CSP model in Section 3.2 and, since we only have two items of value 10 and the target is 100 we need 8 virtual items.

The CSP will now yield all packages that are "just below the target", meaning that if the smallest items from V is added, namely item of value 10, the total value will outreach the target value 100

$$PC_1 = \{\langle E_3, 10\rangle\}$$

$$PC_2 = \{\langle E_2, 1\rangle, \langle E_3, 6\rangle\}$$

$$PC_3 = \{\langle E_2, 2\rangle, \langle E_3, 2\rangle\}$$

$$PC_4 = \{\langle E_1, 1\rangle, \langle E_3, 5\rangle\}$$

$$PC_5 = \{\langle E_1, 1\rangle, \langle E_2, 1\rangle, \langle E_3, 1\rangle\}$$

$$PC_6 = \{\langle E_1, 2\rangle\}$$

For readability, only the equivalence classes E_q with factors $f_q > 0$ have been included above. Now, regarding only the non-zero factors for each equivalence class in the package classes, constraint (B.11) becomes:

$$PC_4 + PC_5 + 2\ PC_6 \geq 2$$

$$PC_2 + 2\ PC_3 + PC_5 \geq 2$$

$$10\ PC_1 + 6\ PC_2 + 2\ PC_3 + 5\ PC_4 + PC_5 \geq 2$$

We only have two items of value 10 and, in PC_4 and PC_5, E_3 has multiplicity 1, while in PC_6 E_3 has multiplicity 2, hence the coefficients in the first inequality. The bin packing requires that all items are placed into bins, therefore there should be an equality sign; when we generated packages though, we used virtual bins also, therefore now we need to allow for more items than actually available in V, but not less, hence the "$\geq$" sign. The same procedure applies to the other two equivalence classes as shown in second and third inequality. Since the problem is a minimization, most of the virtual items will not be used (the ones left are not enough to form another bin), therefore the solution will be optimal with respect to V (as explained in detail later on). Then, the objective is:

$$\min PC_1 + PC_2 + PC_3 + PC_4 + PC_5 + PC6.$$

3.3 Bin Covering

For bin covering we want to maximize the number of bins, and hence it seems to make sense to have bins that are as close to the target (but not below) as possible.

Definition 3: *A bin (or package) is said to be* skinny *if removing from it its least valued item gets it below the target value. That is, a bin b_j is skinny if*

$$B_j - \min(b_j) < t.$$

A feasible solution $f_s = \{b_1, b_2, \ldots, b_m\}$ for a bin covering problem $\langle V, t, \geq \rangle$ is said to be *skinny* if all bins in f_s are skinny, and an arbitrary number of items of V is sorted. The items of V that are not sorted are called the *surplus* items. A feasible solution $f_t = \{b_1, b_2, \ldots, b_m\}$ to a bin covering problem $\langle V, t, \geq \rangle$, is said to be *true* if all items are sorted.

Lemma 2: *For a bin covering problem $\langle V, t, \geq \rangle$, a feasible* true *solution f_t exists if and only a feasible skinny solution f_s exists.*

Proof. Regard a true feasible solution $f_t = \{b_1, b_2, \ldots, b_m\}$. For each non-skinny bin b_j, we can iteratively remove its least valued item $\min(b_j)$ until b_j becomes skinny. Obviously, the resulting skinny solution will have the same number of bins as f_t.

Regard a skinny feasible solution $f_s = \{b_1, b_2, \ldots, b_m\}$. This has a set of non-sorted *surplus* items, V_{sur}. These items can be put on arbitrary skinny bins to make the bins non-skinny. Doing so for a bin $b_j \in f_s$ will increase the value B_j which means that it is still on or above target. Thus, we can from the skinny feasible solution generate a true solution f_t.

For both implications, the number of bins is preserved. □

Theorem 2: *Let B^t_{opt} be an optimal true solution to the bin covering problem $\langle V, t, \geq \rangle$, and let B^s_{opt} be an optimal skinny solution to the same problem. Then*

$$|B^t_{opt}| = |B^s_{opt}|.$$

Proof. If there exists a feasible solution for the skinny problem that is optimal, then no better solution than that exists and, there must exist a feasible solution for the true problem that yields the same result and no better solution can exist. □

3.4 Skinny package generation

As for the bin covering, it is possible to setup a Constraint Satisfaction Problem based on the definition of Fit Bin and run it multiple time to produce, at each iteration, a different valid package, until the problem becomes unfeasible; then we know we have found all feasible packages.

Regard the bin covering problem $\langle V, t, \geq \rangle$, with the equivalence classes E_q $(q = 1, \ldots, p)$. Let f_q $(q = 1, \ldots, p)$ be the factor for E_q, that is, an integer variable representing how many values from E_q that are chosen to form a package class. The CSP formulation is as follows:

$$0 \leq f_q \leq |E_q| \qquad \forall q = 1, \ldots, p \tag{B.16}$$

$$f_q > 0 \Rightarrow \sum_{\substack{j=1 \\ q \neq j}}^{p} \Big(\min(E_j) \cdot f_j + (f_q - 1) \cdot \min(E_q) \Big) < t \qquad \forall q = 1, \ldots, p \tag{B.17}$$

Constraint B.16 limits the factor for each equivalence class to be at most as large as the size of the equivalence class itself; constraint B.17 states that the sum of values contained in a *skinny package* goes below the target value as soon as we reduce the value of any factor by *one*. Unlike the corresponding constraint for the bin packing problem, in this case it is necessary to specify that we can only reduce a value if it is larger than zero.

Let $S = \{f_1^*, \ldots, f_p^*\}$ be the solution of the CSP problem, where f_i^* $\forall i = 1, \cdots, p$ is the factor selected for the equivalence class p, then the additional constraint is:

$$\neg \Big(\bigwedge_{f_i^* \in S} (f_i = f_i^*) \Big) \tag{B.18}$$

Example of Equivalence Class Formulation Consider the bin covering problem $\langle V, t, \geq \rangle$, with V and t as in section 3.2 and so are the equivalence classes.

We can generate four skinny package classes:

$$PC_1 = \{\langle E_1, 2 \rangle\}$$

$$PC_2 = \{\langle E_1, 1 \rangle, \langle E_2, 2 \rangle\}$$

$$PC_3 = \{\langle E_1, 1 \rangle, \langle E_2, 1 \rangle, \langle E_3, 1 \rangle\}$$

$$PC_4 = \{\langle E_2, 2\rangle, \langle E_3, 2\rangle\}$$

For readability, only the equivalence classes E_q with factors $f_q > 0$ have been included above.
Now, looking only at the non-zero factors for each equivalence class in the package classes, constraint (B.11) becomes:

$$2\ PC_1 + PC_2 + PC_3 \leq 2$$

$$2\ PC_2 + PC_3 + 2\ PC_4 \leq 2$$

$$PC_3 + 2\ PC_4 \leq 2$$

Since the the bin covering does not require all items to be allocated to binsl, the equality constraint can be relaxed, hence the inequalities above.

Finally, the objective is:

$$\max PC_1 + PC_2 + PC_3 + PC_4.$$

One optimal solution is to select PC_1 once ($PC_1 = 1$) and PC_4 once ($PC_4 = 1$). However, another optimal solution is to select two "instances" of PC_3 ($PC_3 = 2$). Both of these are of course skinny solutions.

So, we really only need to generate fit (for packing) and skinny (for covering) packages and package classes to get optimal solutions from the subset and equivalence class formulations. This saves a lot of computational effort, as shown in Section 5.

4 Sub-Optimal Solutions

The equivalence class formulation significantly improves the runtime performance of the optimizer compared to the naïve formulation, as shown in Section 5. Nevertheless, BC is still a combinatorial NP-hard problem and for some problems, calculating the true optimum might be expensive in terms of computational effort. The alternative is to consider heuristic methods that can provide a sub-optimal solution in a shorter time. Based on the equivalence classes approach, a heuristic method was developed to simplify the problem and calculate a suboptimal solution faster. One hypothesis that led to the heuristic is that the number of package classes related to a BC problem $\langle V, t, \geq\rangle$ does not depend entirely on $|V|$, but rather on the number of

equivalence classes and their cardinality; the more different values, the more possible combinations, the more package classes.

Therefore, the goal of such method is to provide a set of approximated equivalence classes, C^*, where a certain number of consecutive exact equivalence classes are merged together. We call the approximated equivalence classes *chains*.

Let $E = \langle E_1, E_2, \ldots, E_p \rangle$ be the set of p number of equivalence classes over the values of V, ordered so that $\min(E_i) < \min(E_{i+1})$ (for $i = 1, \ldots, p-1$). Let C be a set of *chains*, sets of one or more consecutive equivalence classes from E, $C = \bigcup_{i=1}^{p-l+1} \{E_i, E_{i+1}, \ldots E_{i+l-1}\}$ for $l = 1, \ldots, p$, and let the size of a chain be the sum of the sizes of the equivalence classes it is composed of; for $C_j \in C$, $\|C_j\| = \sum_{E_i \in C_j} |E_i|$.

Note that chains will have common elements (equivalence classes), just as packages, therefore it is possible to define the set O_i of all chains containing a certain value.

Example of Chains Generation

$E = \langle E_1, E_2, E_3, E_4 \rangle$, $p = 4$, with $\min(E_i) < \min(E_{i+1})$ for $i = 1, \ldots, 3$,

$C = \bigcup_{i=1}^{p-l+1} \{E_i, E_{i+1}, \ldots E_{i+l-1}\}$ for $l = 1, \ldots, p$

$$
\begin{array}{ll}
\{E_1\}, \{E_2\}, \{E_3\}, \{E_4\}, & l = 1, i = 1, \ldots, p \\
\{E_1, E_2\}, \{E_2, E_3\}, \{E_3, E_4\}, & l = 2, i = 1, \ldots, p-1 \\
\{E_1, E_2, E_3\}, \{E_2, E_3, E_4\}, & l = 3, i = 1, \ldots, p-2 \\
\{E_1, E_2, E_3, E_4\} & l = 4, i = 1, \ldots, p-3
\end{array}
$$

4.1 Heuristic for the bin packing problem

When it comes to the bin packing problem, a way to generate the above mentioned chains is to merge exact equivalence classes as explained in the above section and to assign to all the values of the resulting approximated equivalence class the largest value of all classes that have been merged. This is done so that the solution generated when solving the simplified problem is valid: if a value smaller than the one belonging to the largest class merged were to be assigned to the approximated class, the optimal solution to the simplified problem might be smaller than the optimal solution to the original

problem and, therefore, unfeasible.

Definition 4: *Let define the* minimum theoretical number of bins M *as the number of bins we could achieve if we could break down the items into smaller ones and reallocate the overweight from each bin to form other ones:* $M = {}^{W}/_{t}$. *Such value can be achieved by relaxing the integrality constraint in the initial problem, as pointed out in [11] where such value is defined as* lower bound.

Let the chain E_{l-m} be the result of merging the equivalence classes E_l, E_m, where $\min(E_l) < \min(E_m)$. Then $\min(E_{l-m}) = \min(E_m), |E_{l-m}| = |E_l| + |E_m|$.

$$|E_{l-m}| \cdot \min(E_{l-m}) = (|E_l| + |E_m|) \cdot \min(E_m) > \\ |E_l| \cdot \min(E_l) + |E_m| \cdot \min(E_m)$$

The gain γ is then:

$$\min(E_m) \cdot (|E_l| + |E_m|) - \\ |E_l| \cdot \min(E_l) + |E_m| \cdot \min(E_m) = \\ |E_l| \cdot (\min(E_m) - \min(E_l))$$

The new minimum theoretical number of bins is $M' = {}^{(W+\gamma)}/_{t}$. The gain can be used to decide which classes is better to merge when simplifying an instance. Using a greedy algorithm it is possible to quickly generate all chains and calculate γ for each of them. It is then possible to formulate a MILP model to select the chains that produce the minimum loss, given a desired number of equivalence classes d.

Let x_i $(i = 1, \ldots, k)$ be 0-1-variables representing whether the chain C_i is

chosen ($x_i = 1$) or not ($x_i = 0$). The MILP formulation is as follows:

$$\min \sum_{i=1}^{k} x_i \cdot \gamma_i \tag{B.19}$$

$$\sum_{i=1}^{p} x_i = d \tag{B.20}$$

$$\sum_{x_i \in O_j} x_i = 1 \qquad \forall j = 1, \ldots, p \tag{B.21}$$

$$x_i \in \{0, 1\} \qquad \forall i = 1, \ldots, k \tag{B.22}$$

The objective function (B.19) is the sum of all gains for the chains that are chosen. Constraint B.20 states that exactly d chains have to be chosen. Constraint (B.21) states that overlapping chains are mutually exclusive. Finally, (B.22) sets the variable domains to be binary.

Note that the heuristic has been developed bearing in mind that, with a normal distribution, there are only a few values at the edges of the *bell curve*, while most of the values appear in the middle of it. Therefore, while containing the same number of values, the chains that are closer to the edges will contain more classes from E, involving a larger loss than the ones closer to the centre. However, as those values are smaller in number compared to the ones in the middle, the overall loss seems not to be significant.

4.2 Heuristic for the bin covering problem

Once again it is possible to draw inspiration from the results achieved for the bin packing problem to develop a working heuristic method for the bin covering problem too. By merging equivalence classes it is in fact possible to generate a simplified problem that provides a sub optimal solution. This time it is required to assign to the resulting equivalence class the value of the items from the class with the smallest values, in order to generate a valid solution. Since this is a maximization problem, it makes more sense to talk about *loss* σ (rather than a gain) that affects the maximum (instead of minimum) theoretical number of bins. Once again this value corresponds to the optimum of the cost function for the bin covering when relaxing the integrality constraint.

It is possible to calculate the *loss* σ related to the merging of two or more equivalence classes in terms of decrease in the total value W and, therefore,

of the maximum theoretical number of bins M.

Let the chain E_{l-m} be the result of merging the equivalence classes E_l, E_m, where $\min(E_l) < \min(E_m))$. Then $\min(E_{l-m}) = \min(E_l), |E_{l-m}| = |E_l| + |E_m|$.

$$|E_l| \cdot \min(E_l) + |E_m| \cdot \min(E_m) > \\ |E_{l-m}| \cdot \min(E_{l-m}) = \min(E_l) \cdot (|E_l| + |E_m|)$$

The loss σ is then:

$$|E_l| \cdot \min(E_l) + |E_m| \cdot \min(E_m) - \\ \min(E_l) \cdot (|E_l| + |E_m|) = \\ (\min(E_m) - \min(E_l) \cdot |E_m|)$$

The new maximum theoretical number of bins is $M' = {}^{(W-\sigma)}/_{t}$. Setting up exactly the same MILP model as in 4.1 it is possible to find the set of merged classes that minimizes the gain while guaranteeing a desired number of equivalence classes for the simplified problem.

Example of Chain Optimization

Consider the example shown in Section 4, and assume that the values and sizes of the equivalence classes are respectively $\min(E_1) = 13$, $|E_1| = 10$, $\min(E_2) = 15$, $|E_2| = 7$, $\min(E_3) = 20$, $|E_3| = 12$, $\min(E_4)$, $|E_4| = 3$. If the desired number of classes is $d = 2$ we can compute the loss for each chain based on (B.19)–(B.22):

$$C_1 = \{E_1\} \qquad \sigma(C_1) = 0$$
$$C_2 = \{E_2\}, \qquad \sigma(C_2) = 0$$
$$C_3 = \{E_3\}, \qquad \sigma(C_3) = 0$$
$$C_4 = \{E_4\}, \qquad \sigma(C_4) = 0$$
$$C_5 = \{E_1, E_2\}, \qquad \sigma(C_5) = 7 \cdot (15 - 13) = 14$$
$$C_6 = \{E_2, E_3\}, \qquad \sigma(C_6) = 12 \cdot (20 - 15) = 60$$
$$C_7 = \{E_3, E_4\}, \qquad \sigma(C_7) = 3 \cdot (25 - 20) = 15$$

$$
\begin{aligned}
C_8 &= \{E_1, E_2, E_3\}, \\
&\qquad \sigma(C_8) = \sigma(C_5) + 12 \cdot (20 - 13) = 98 \\
C_9 &= \{E_2, E_3, E_4\}, \\
&\qquad \sigma(C_9) = \sigma(C_6) + 3 \cdot (25 - 15) = 90 \\
C_{10} &= \{E_1, E_2, E_3, E_4\}, \\
&\qquad \sigma(C_{10}) = \sigma(C_8) + 3 \cdot (25 - 13) = 134
\end{aligned}
$$

According to constraint (B.20) only d chains can be selected:

$$C_1 + C_2 + C_3 + C_4 + C_5 + C_6 + C_7 + C_8 + C_9 + C_{10} = 2$$

According to constraint (B.21) some chains are mutually exclusive, so each equivalence class must be picked exactly once. As stated before, chains are sets, but with some abuse of notation, we here use them as binary variables to state whether a chain is selected ($C_i = 1$) or not ($C_i = 0$).

$$
\begin{aligned}
&C_1 + C_5 + C_{10} = 1 \\
&C_2 + C_5 + C_6 + C_8 + C_9 + C_{10} = 1 \\
&C_3 + C_6 + C_7 + C_8 + C_9 + C_{10} = 1 \\
&C_4 + C_7 + C_9 + C_{10} = 1
\end{aligned}
$$

Finally, the objective function to minimize is:

$$
\begin{aligned}
&C_1 \cdot \sigma(C_1) + C_2 \cdot \sigma(C_2) + C_3 \cdot \sigma(C_3) \\
&+C_4 \cdot \sigma(C_4) + C_5 \cdot \sigma(C_5) + C_6 \cdot \sigma(C_6) + C_7 \cdot \sigma(C_7) \\
&+C_8 \cdot \sigma(C_8) + C_9 \cdot \sigma(C_9) + C_{10} \cdot \sigma(C_{10})
\end{aligned}
$$

5 Computational Analysis

We ran an extensive analysis over 1500 generated problems, comparing both the standard and the equivalence class formulations for both the bin packing and bin covering problem; we also compare the standard and equivalence classes formulations for the bin packing problem over different benchmark sets from the literature:

Table 1: Comparison of the standard formulation and the equivalence class formulation over the benchmark instance sets, showing the number of solved instances within one minute and the average time calculated over the instances that did not time out. For each instance set it is reported the total number of instances and the number of instances that generate less than ten million packages. The '-' is used when no instance is solved, while the '*' means that the package generation algorithm run out of memory.

	Instances		STD		EQU	
	Complete Set	$\leq 1.0 \times 10^7$	solved	average	solved	average
Falkenauer U	80	80	16	34.11	80	1.64
Falkenauer T	80	80	10	30.07	80	0.76
Scholl 1	720	313	185	26.92	294	1.29
Scholl 2	480	54	50	6.86	48	7.69
Scholl 3	10	7	0	-	0	-
Schwerin 1	100	100	51	32.23	26	48.49
Schwerin 2	100	100	40	37.72	5	53.66
Wäscher	17	0	0	-	*	*
Schoenfield	28	0	0	-	*	*

- Falkenauer [14]: two sets with 80 instances each;
- Scholl [15] three sets with 720, 480 and 10 instances respectively;
- Schwerin [16] two sets with 100 instances each;
- Wäscher [17] a set with 17 instances;
- Schoenfield [18] a set with 28 instances.

All instances have been solved using the state-of-the-art MILP solver Gurobi 9 [19]. The time limit is 1200 seconds and the solver has been used with its default setting. All the experiments were performed on an *Intel Core i7 6700K, 4.0 GHZ, 32GB RAM* running *Ubuntu-16.04*. The implementation of all the models presented in this paper, as well as the benchmark instances. The instance generator is available on `https://github.com/sabinoroselli/bin_covering-packing.git`

To generate the instances we approximated a normal distribution. The parameters to generate instances are the number of items, the range of values, the average value of such range (which is the mean of the distribution) and the standard deviation. Since the results show a skewed distribution, we reported the median and upper and lower quartile for each category.

The first set of tests, reported in Table 2 has been run using the standard formulation to solve both the covering and the packing problem. The number of items ranges from 10 to 70 and the target value ranges from 300 to 900. The values of the items range from 130 to 170 and the value of the deviation ranges from 10 to 90; finally, five different instances are generated for each set of parameters by changing the random generation seed. Results show that different values for the deviation do not affect the running time significantly so they are not shown explicitly in the table: for each size and target value, the average over 25 instances is shown (five different values of deviation times five different random seeds).

Results from this first simulation show that, as expected, an increasing number of items makes the problem harder to solve. It is interesting though, to notice that even for relatively high number of items, there are still many instances that can be solved almost instantly which means that also large problems can have trivial solutions. What does affect the complexity of the instance, according to the results, is the target value t; though there are some outliers, most of the unsolved instances for the covering problem have a target value of 500, while for the packing problem it is 700.

In the second set of tests the number of items ranges from 60 to 500, while the other parameters are the same as in the previous tests. For the instance classes counting 200 to 500 items and with a target value of 800 and 900 it was only possible to solve one instance, therefore it was not possible to calculate the quartiles. The results, summarized in Table 3 show, as for the tests run for the standard formulation, an increasing amount of time required to find the optimal as the number of items and the target value increases (given a certain distribution and, thus, a certain number of equivalence classes). Unlike the standard approach though, the equivalence classes formulation seems to have a more steady trend: the former's performance is heavily affected by the intrinsic hardness of a specific instance, being able sometimes to immediately solve large size instances with a large target value, while getting stuck on relatively small sized instances; on the other hand, the latter's behaviour is more predictable and steadily increases with the instances parameters, as shown in Figure 1. Also, the comparison of Table 2 and Table 3 show tighter values of the quartiles for the equivalence classes formulation; this mean that the solving time for a given class of instances (in terms of generated packages for example) is more predictable.

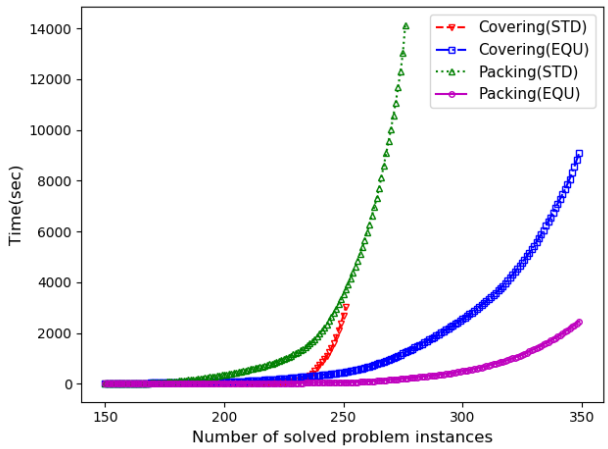

Figure 1: Comparison of model formulations (STD stands for *standard* and EQU for *equivalence classes*) and sorting type over generated problems.

Another conclusion we can draw from the data is the strong correlation between the solving time and the number of packages, which in turn is affected by the target value and, to a lesser extent, by the number of objects. A larger target value means an exponentially higher number of possible combinations to form skinny (or fit) packages; also having a higher number of items in each equivalence class means that it is possible to form more combinations of them that make valid packages, though this is true only up to a certain value. For instance, if the target value is 300, having two or 200 items of value 200 does not make any difference.

For this reason, an instance with 60 items and a target value of 300 only yields a few thousand packages, while the same instance with a target value of 900 can count millions of them. As mentioned before, the increase in the number of items also causes an increase in the number of packages, which seems nonetheless to happen within the same order of magnitude.

Though the number of packages has a direct impact on the solving time, the equivalence class formulation still proves to be efficient to solve very large sized problems, being scalable in terms of number of items (which is usually the limitation for the standard approach). Even so, the package generation time via a greedy algorithm and the model generation time are directly proportional

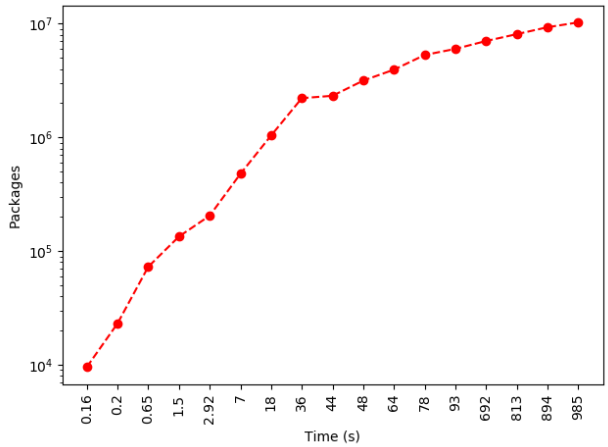

Figure 2: Evaluation of package generation time (in seconds) against number of generated packages (log scale).

to the number of generated packages and for very hard instances they might not be negligible. Figure 2 shows how the package generation time increases with respect to the number of packages (benchmark instances from the *Scholl* set have been employed to evaluate the package generation efficiency): it is still close to one minute for instances leading to four million packages but it goes up to about fifteen minutes for instances of around ten million packages. Though we have not investigated the subject for this work, we believe that there is quite some room for improvement in the implementation of the package generation algorithm (changing the programming language to begin with, since now it is written in Python). While solving the benchmark instances, the largest instances we could solve before running out of memory counted circa thirty million packages; memory overflow is another matter we plan to address in our future work.

The third set of instances, summarized in Table 1, shows the performance of the equivalence classes method over different sets of benchmark instances. In order to compare our formulation to the other relevant algorithms we found in the literature, we refer to Table I and Table II of [3], where such algorithms are evaluated using the same benchmark sets listed above, and setting the time limit to one minute (the implementation for such algorithms is available at the

authors web page [20]. Though the computers used are different, we believe that the comparison still gives a hint of the methods potentials. Nonetheless, we decided to run the benchmark instances again for the standard formulation (called *Basic ILP* in [3], since the authors used a different solver). Moreover, the algorithms listed in those tables, are specifically tailored to solve the bin packing problem, while our formulation provides a linear program that can be fed to any solver (or extended with additional constraints) and so can be done with the standard formulation. Therefore we decided to make a more rigorous comparison between the standard formulation and the equivalence classes formulation.

As already discussed when commenting on Table 3, the number of generated packages directly affects the solving time when using the equivalence classes approach; therefore we only considered instances counting less than ten millions packages. Table 1 shows the number of instances in each set and, next to it, the number of instances that lead to generate less than ten million packages. The equivalence class formulation is much faster than the standard formulation for the *Falkenauer* sets and can deal with all instances in less than 2 seconds each. When it comes to the *Scholl* instances, in the first set the equivalence classes formulation is still much faster and can deal with almost twice as many instances before the time limit while it performs very similarly, though slightly worse in the second set. Neither method can deal with any of the instances in the third set. The standard formulation performs considerably better than the equivalence classes one in both *Schwerin* sets, both in terms of instances solved and average time. Finally, the standard formulation cannot solve any of the instances from either the *Wäscher* or the *Schoenfield* set within the time limit, while the equivalence class formulation cannot even get started, since the computer ran out of memory while generating the packages.

A possible remedy to handle such hard instances is to reduce the number of classes by merging them into chains as explained in sections 4.2 and 4.1. The heuristic does not guarantee an optimal value to the original problem, but it can drastically reduce the number of packages, thus speeding up the model generation tremendously, while producing very close-to-optimal results. Table 4 shows an example for both the covering and the packing problem where the number of equivalence classes is progressively reduced from the original one, shown on the first line (the one that would lead to the true optimal

solution). The instance is evaluated considering two different target values: 450 is an exact multiple of 150 (the mean value of the items) while 525 is as far as possible from being an exact multiple. Also, two different values for the deviation are selected: 10 (corresponding to the data shown in the upper part of Table 4) and 90 (corresponding to the remaining data); neither of the two parameters seems to largely affect the accuracy. What we can see though, is a dramatic reduction in the number of generated packages as the number of classes decreases, while the optimal value for the simplified instances is still very close to the optimum for the original one.

Acknowledgements

The authors thank Folkmar Frederik Ramcke for implementing the algorithms for package generation.

6 Conclusions

In this paper we have presented alternative formulations to solve both the bin covering and the bin packing problem; we have shown that these formulations perform particularly well when the number of different values in the problem instance is limited. In such cases, our formulations allow to to solve problem counting hundreds of items in a considerably short time. This feature can prove useful for industrial applications where the number of items is high but the range of values is limited, such as battery recycling (for bin packing) or fixed tray weight sorting in food processing (bin covering). When this is not the case, our formulations allow for problem simplification by means of merging equivalence classes that still provides a close to optimal solution, while dramatically reducing the computation time. Moreover, the concept of skinny/fit packages can constitute a solid base to improve existing specific-purpose algorithms such as those given by [3], which we intend investigate further.

Table 2: Set of generated instances solved using the standard formulation for both the packing and the covering problem. The size of the instances ranges from 10 to 70 items and the target value from 300 to 900. The median time is reported (calculated over the solved instances) as well as the lower and upper quartile and the number of instances that timed out. The timeout is set to 1200 seconds. If no instance for a given category is solved, the cell is marked with the symbol '-'.

COVERING STANDARD

	300				400				500				600				700				800				900			
	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO
10	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0
20	0.00	0.00	0.02	0	0.00	0.00	0.00	0	0.00	0.00	0.02	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0
30	0.01	0.01	1.16	4	0.01	0.01	0.01	0	8.59	0.09	9.58	4	0.01	0.01	0.01	0	0.01	0.01	0.01	0	0.01	0.01	0.01	0	0.01	0.01	0.02	0
40	0.06	0.02	0.16	7	0.01	0.01	0.01	0	18.57	13.95	22.74	12	0.01	0.01	0.02	0	0.01	0.01	0.01	0	0.04	0.01	0.08	0	0.01	0.01	0.01	0
50	0.07	0.03	0.15	3	0.46	0.35	0.54	0	35.50	26.22	54.81	11	0.02	0.02	0.02	0	0.01	0.01	0.02	0	0.42	0.12	24.89	6	0.02	0.01	0.02	0
60	0.17	0.12	0.27	15	0.03	0.03	0.03	0	90.25	32.75	116.26	13	0.03	0.02	0.03	0	0.03	0.03	0.10	0	-	-	-	25	0.02	0.02	0.03	0
70	0.17	0.15	0.25	6	0.90	0.04	2.74	0	65.83	12.89	236.56	11	0.04	0.03	0.04	0	0.03	0.03	0.07	4	-	-	-	25	0.03	0.03	0.03	0

PACKING STANDARD

	300				400				500				600				700				800				900			
	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO
10	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0
20	0.00	0.01	0.10	0	0.05	0.03	0.08	0	0.00	0.00	0.03	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0
30	0.01	0.01	0.23	0	0.18	0.12	0.26	0	0.08	0.00	0.14	0	0.00	0.00	0.01	0	0.01	0.01	0.04	0	0.00	0.00	0.00	0	0.00	0.00	0.02	0
40	3.85	0.40	30.68	0	0.76	0.49	1.98	0	1.13	0.33	2.09	0	0.01	0.01	0.06	0	1.95	0.12	5.14	15	0.01	0.01	0.01	0	0.01	0.01	0.01	0
50	18.33	8.97	82.04	6	6.76	1.15	21.46	0	3.75	1.00	4.95	0	0.02	0.02	0.02	4	4.56	3.06	9.69	16	0.01	0.01	0.01	0	0.01	0.01	0.01	0
60	39.58	0.10	170.01	11	21.77	3.08	40.42	7	7.61	5.44	13.96	0	0.07	0.04	0.22	0	79.61	18.25	124.53	15	0.01	0.01	0.01	0	0.05	0.03	0.16	0
70	1.02	0.10	44.92	13	473.65	53.34	560.20	8	19.92	16.06	27.06	0	0.04	0.04	0.04	3	153.92	20.90	343.86	13	0.02	0.02	0.02	0	0.02	0.02	0.03	4

Table 3: Set of generated instances solved using the equivalence classes formulation for both the packing and the covering problem. The size of the instances ranges form 60 to 500 items and the target value ranges from 300 to 900. The median time is reported as well as the lower and upper quartile and average number of packages generated (up to tens of thousands, numbers are shown in regular notation, afterwards the scientific notation is employed. The timeout is set to 1200 seconds. The symbol '-' means that only one instance for that class was solved, therefore it was not possible to calculate the upper or lower quartile.

COVERING EQUIVALENCE CLASSES																												
	300			400			500			600			700			800			900									
	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack
60	0.01	0.01	0.02	1727	0.06	0.05	0.06	5534	0.38	0.29	0.44	44566	1.02	0.79	1.38	1×10^5	7.39	3.91	8.73	3×10^5	39.30	29.96	56.11	1×10^6	106.04	65.85	144.22	3×10^6
70	0.02	0.01	0.02	1948	0.06	0.05	0.07	6321	0.46	0.37	0.52	53643	1.34	0.97	1.96	1×10^5	8.51	6.39	10.85	4×10^5	53.79	41.79	68.32	2×10^6	158.50	110.38	210.63	5×10^6
80	0.02	0.02	0.02	2210	0.07	0.05	0.07	7294	0.57	0.47	0.64	65301	1.94	1.75	2.31	2×10^5	12.48	7.52	13.51	5×10^5	82.53	69.75	98.70	3×10^6	266.01	224.57	359.33	6×10^6
100	0.02	0.02	0.03	2655	0.07	0.06	0.07	8652	0.80	0.66	0.86	82794	3.03	2.37	3.56	2×10^5	18.54	14.43	23.35	7×10^5	160.79	127.88	178.73	4×10^6	462.48	376.44	517.59	1×10^7
150	0.03	0.02	0.03	3128	0.09	0.08	0.10	10261	0.97	0.78	1.11	1×10^5	5.28	4.17	5.55	3×10^5	26.48	19.24	33.50	9×10^5	215.04	121.05	248.88	5×10^6	679.03	-	-	2×10^7
200	0.02	0.02	0.03	3340	0.10	0.09	0.10	10936	0.99	0.91	1.01	1×10^5	5.30	4.73	5.77	4×10^5	27.39	24.41	29.01	1×10^6	324.04	-	-	7×10^6	1115.65	-	-	2×10^7
300	0.01	0.01	0.02	3432	0.09	0.08	0.10	11330	0.96	0.93	1.01	1×10^5	6.04	5.63	6.96	4×10^5	27.71	27.07	29.18	1×10^6	301.28	-	-	8×10^6	1084.21	-	-	2×10^7
500	0.01	0.01	0.01	3477	0.06	0.05	0.08	11469	0.80	0.77	0.90	1×10^5	5.66	5.15	6.02	4×10^5	27.48	25.90	31.20	1×10^6	356.01	-	-	8×10^6	1109.46	-	-	2×10^7

PACKING EQUIVALENCE CLASSES																												
	300			400			500			600			700			800			900									
	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack
60	0.00	0.00	0.00	281	0.00	0.00	0.00	524	0.03	0.02	0.03	5555	0.33	0.25	0.38	29853	0.64	0.47	0.81	50518	8.01	4.83	15.86	3×10^5	32.80	17.17	48.84	1×10^6
70	0.00	0.00	0.00	303	0.00	0.00	0.01	569	0.04	0.03	0.04	6327	0.37	0.29	0.45	30505	0.73	0.67	0.86	60849	9.56	8.01	10.34	4×10^5	43.81	25.19	64.88	1×10^6
80	0.00	0.00	0.00	329	0.00	0.00	0.01	625	0.04	0.04	0.04	7299	0.48	0.36	0.56	36765	0.96	0.89	1.10	74909	13.38	10.32	15.22	5×10^5	73.35	45.26	81.95	2×10^6
100	0.00	0.00	0.00	370	0.01	0.01	0.01	699	0.05	0.04	0.06	8644	0.49	0.38	0.61	46854	1.19	1.10	1.80	1×10^5	18.61	15.15	33.61	6×10^5	102.59	86.19	140.68	3×10^6
150	0.00	0.00	0.00	412	0.01	0.01	0.01	786	0.04	0.04	0.05	10237	0.56	0.49	0.62	58950	1.47	0.96	3.07	1×10^5	21.64	15.06	26.16	9×10^5	145.84	106.05	182.35	3×10^6
200	0.00	0.00	0.00	429	0.01	0.01	0.01	821	0.07	0.06	0.08	10904	0.60	0.53	0.79	64416	1.82	1.56	3.59	1×10^5	26.50	-	-	1×10^6	162.30	-	-	5×10^6
300	0.00	0.00	0.00	436	0.01	0.01	0.01	841	0.05	0.04	0.05	11286	0.71	0.59	0.81	67163	2.59	1.64	3.16	1×10^5	304.40	-	-	1×10^6	162.70	-	-	5×10^6
500	0.00	0.00	0.00	440	0.01	0.01	0.01	849	0.09	0.08	0.10	11417	0.73	0.61	0.89	68213	2.81	2.17	4.46	2×10^5	673.20	-	-	1×10^6	154.10	-	-	5×10^6

Table 4: Comparison of the optimal solution and solving time (in seconds) with respect to the number of generated packages for an instance of 200 items solved with both packing and covering method when varying the number of classes (Cl.) by simplifying the original instance. The instance is evaluated for target values of 450 and 525 and deviation values of 10 (top part of the table) and 90 (bottom part).

Cl.	COVERING						PACKING					
	Standard Deviation: 10											
	450			525			450			525		
	Packages	Optimum	Time	Packages	Optimum	Time	Packages	Optimum	Time	Packages	Optimum	Time
45	62999	65	0.47	181800	50	2.10	8964	67	0.08	15969	65	0.26
40	44616	65	0.35	118894	50	1.43	6768	67	0.07	11525	65	0.20
35	26057	65	0.19	72368	50	0.50	4437	67	0.04	7814	65	0.12
30	15931	65	0.10	39935	50	0.28	2486	67	0.03	4955	66	0.03
25	8154	65	0.08	19788	50	0.13	1566	67	0.02	2925	66	0.02
20	3742	65	0.03	8621	50	0.05	807	67	0.00	1540	66	0.01
15	1512	65	0.01	3061	50	0.02	411	67	0.00	681	66	0.00
10	445	64	0.00	715	50	0.00	106	68	0.00	220	66	0.00
	Standard Deviation: 90											
	450			525			450			525		
	Packages	Optimum	Time	Packages	Optimum	Time	Packages	Optimum	Time	Packages	Optimum	Time
80	540384	66	9.09	2151942	56	53.90	47068	67	0.52	219327	57	3.77
75	438155	66	6.40	1683682	56	41.53	38926	67	0.37	173769	57	5.44
70	315242	65	3.77	1222000	56	33.51	33502	67	0.25	143324	57	2.24
65	214423	65	1.89	825952	56	15.09	25168	67	0.18	99023	57	6.27
60	181567	65	1.90	647241	56	13.17	18370	67	0.09	67044	57	2.02
55	130228	65	1.26	450641	56	8.11	14857	67	0.08	52659	57	1.19
50	82868	65	0.69	293594	56	2.56	10848	67	0.07	36831	57	0.73
45	67720	65	0.44	216105	56	3.28	7912	67	0.07	25138	57	0.47
40	40267	65	0.23	128418	56	1.95	5988	67	0.04	18789	57	0.16
35	25224	65	0.19	77416	56	0.76	3583	67	0.02	10421	57	0.21
30	14716	65	0.09	42137	56	0.46	2579	67	0.02	7217	57	0.13
25	7903	65	0.05	21239	56	0.26	1491	67	0.02	3650	57	0.03
20	3635	65	0.03	8834	56	0.05	826	67	0.01	1904	58	0.01
15	1437	65	0.01	3149	55	0.02	310	68	0.00	674	58	0.01
10	344	64	0.00	677	54	0.00	112	69	0.00	213	59	0.00

References

[1] E. G. Coffman, J. Csirik, G. Galambos, S. Martello, and D. Vigo, "Bin packing approximation algorithms: Survey and classification," in *Handbook of combinatorial optimization*, Springer New York, 2013, pp. 455–531.

[2] S. F. Assmann, D. S. Johnson, D. J. Kleitman, and J.-T. Leung, "On a dual version of the one-dimensional bin packing problem," *Journal of algorithms*, vol. 5, no. 4, pp. 502–525, 1984.

[3] M. Delorme, M. Iori, and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," *European Journal of Operational Research*, vol. 255, no. 1, pp. 1–20, 2016.

[4] G. Belov and G. Scheithauer, "A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting," *European Journal of Operational Research*, vol. 171, no. 1, pp. 85–106, 2006.

[5] F. Brandão and J. P. Pedroso, "Bin packing and related problems: General arc-flow formulation with graph compression," *Computers & Operations Research*, vol. 69, pp. 56–67, 2016.

[6] F. Clautiaux, S. Hanafi, R. Macedo, M.-É. Voge, and C. Alves, "Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints," *European Journal of Operational Research*, vol. 258, no. 2, pp. 467–477, 2017, ISSN: 0377-2217.

[7] M. Delorme and M. Iori, "Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems," *INFORMS Journal on Computing*, vol. 32, no. 1, pp. 101–119, 2020.

[8] M. Dell'Amico, F. Furini, and M. Iori, "A branch-and-price algorithm for the temporal bin packing problem," *Computers & Operations Research*, vol. 114, pp. 1–16, 2020, ISSN: 0305-0548.

[9] S. Roselli, F. Hagebring, S. Riazi, M. Fabian, and K. Åkesson, "On the use of equivalence classes for optimal and sub-optimal bin covering," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2019, pp. 1004–1009.

[10] L. V. Kantorovich, "Mathematical methods of organizing and planning production," *Management science*, vol. 6, no. 4, pp. 366–422, 1960.

[11] M. Labbé, G. Laporte, and S. Martello, "An exact algorithm for the dual bin packing problem," *Operations Research Letters*, vol. 17, no. 1, pp. 9–18, 1995.

[12] M. Peeters and Z. Degraeve, "Branch-and-price algorithms for the dual bin packing and maximum cardinality bin packing problem," *European Journal of Operational Research*, vol. 170, no. 2, pp. 416–439, 2006.

[13] M. J. Brusco, H. F. Köhn, and D. Steinley, "Exact and approximate methods for a one-dimensional minimax bin-packing problem," *Annals of Operations Research*, vol. 206, no. 1, pp. 611–626, Jul. 2013.

[14] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *Journal of heuristics*, vol. 2, no. 1, pp. 5–30, 1996.

[15] A. Scholl, R. Klein, and C. Jürgens, "Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem," *Computers & Operations Research*, vol. 24, no. 7, pp. 627–645, 1997.

[16] P. Schwerin and G. Wäscher, "The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP," *International Transactions in Operational Research*, vol. 4, no. 5-6, pp. 377–389, 1997.

[17] G. Wäscher and T. Gau, "Heuristics for the integer one-dimensional cutting stock problem: A computational study," *Operations-Research-Spektrum*, vol. 18, no. 3, pp. 131–144, 1996.

[18] J. E. Schoenfield, "Fast, exact solution of open bin packing problems without linear programming. draft," *US Army Space & Missile Defence Command, Huntsville*, vol. 20, p. 72, 2002.

[19] Gurobi Optimization, LLC, *Gurobi optimizer reference manual, version 9*, 2020.

[20] M. Delorme, M. Iori, and S. Martello, "BPPLIB: A library for bin packing and cutting stock problems," *Optimization Letters*, vol. 12, no. 2, pp. 235–250, 2018.

PAPER C

Solving the Conflict-Free Electric Vehicle Routing Problem Using SMT Solvers

Sabino Francesco Roselli, Martin Fabian, Knut Åkesson

Proceedings of 2021 29th Mediterranean Conference on Control and Automation (MED),
pp. 542–547, Dec. 2018.
 DOI: 10.1109/MED51440.2021.9480202

The layout has been revised.

Abstract

The vehicle routing problem is a combinatorial optimization problem of computing routes to serve customers while minimizing a cost function, typically the traveled distance or the number of vehicles required. Industrial applications of the problem in manufacturing plants are the scheduling and routing of Automated Guided Vehicles (AGVs) to deliver material between storage areas and assembly stations. For in-plant transportation it is necessary to take the limited space of the plant floor into account during scheduling and routing in order to limit the number of AGVs that are at certain areas at a given time. In addition, AGVs are most often powered by batteries and therefore have limited operating range and non-negligible charging time that will also affect the scheduling and routing decisions. In this paper we provide a monolithic model formulation for the scheduling and routing of AGVs with given time-windows for delivering material, restricted by capacity constraints on the path network, and with the need for battery recharge. The problem is modelled and solved using optimizing Satisfiability Modulo Theory (SMT) solvers. The approach is evaluated on a set of generated problem instances, showing that the solver can handle medium size instances in a reasonable amount of time.

1 Introduction

Nowadays automation is increasingly used for material handling and delivery [1] and many companies are implementing automated, *just-in-time* delivery systems [2]. One option is to deploy a fleet of automated guided vehicles (AGVs) that pick up and deliver material and components to the due workstation just-in-time for being used. Just-in-time delivery benefits from using optimization to solve the scheduling and routing of the AGVs in order to minimize the traveled-distance and using as few vehicles as necessary while still delivering all material within the given time-windows.

The standard vehicle routing problem (VRP) [3] is the combinatorial opti-

mization problem of designing routes for vehicles to serve customers so that a cost function, typically either the number of vehicles or the total travelled distance, or a combination of them, is minimized. As the delivery of components to the assembly line has to be done just-in-time, vehicles cannot arrive too late or too early to the workstation. The vehicle routing problem with time windows (VRPTW) [4] is a variation of the standard VRP that accounts for these additional constraints. AGVs are often powered by batteries with limited capacity and thus need to recharge during service. The electric vehicle routing problem (E-VRPTW) [5] is a variant of the standard VRP that accounts for limited operating range of the vehicles and the use of charging stations. Another practical aspect of the fleet management system to consider, is the capacity of the road segments, i.e. the simultaneous number of vehicles a segment can accommodate. For example, if a segment is too small, multiple vehicles cannot operate in it simultaneously without the time used to travel through that segment being significantly longer than if fewer vehicles were operating in the same segment. As for the previous cases, there exists a variation of the standard VRP that accounts for limited capacity of the road segments; the dispatch and conflict-free routing problem (DCFRP) [6]. Recent contributions on the DCFRP are presented in [7], [8].

To our best knowledge, the combined problem of VRP with time-windows, the need to recharge vehicles and taking the limited capacity constraints have not been defined and solved previously. In this paper we therefore define the *conflict-free electric vehicle routing problem* (CF-EVRP) where we address all these features in one problem definition. We consider the cost function to be defined by the total traveled distance although other cost criteria can also be used.

For all the previously defined variants of the VRP problem, there exist Mixed Integer Linear Programming (MILP) models. Using MILP to handle the problem offers high flexibility in terms of extending the model as new constraints are required. On the other hand, the computational complexity increases as the model grows (in terms of number of customers and/or vehicles and, therefore, variables and constraints), especially when the new requirements involve combinatorial logical constraints. For this reason MILP solvers do not scale well to larger problems and specific-purpose algorithms involving local search [9], column generation [10] or stochastic methods [11], [12] are used instead. However, these methods are typically tailor-made for

solving specific types of problems and extending them to new problem variants might not be possible or non-trivial. The need to add constraints that are not part of the textbook problems, result in MILP still being used for industrial applications, or simple heuristic rules are used when MILP fails to scale.

In our previous works [13], [14] we showed that optimizing Satisfiability Modulo Theory (SMT, [15], [16]) solvers outperformed MILP solvers on combinatorial scheduling problems (job-shop problems) involving many logical constraints. The natural abilities of SMT solvers to handle combinatorial constraints natively make it an interesting candidate for solving CF-EVRP problems.

The contributions of this paper are: (i) Designing a model for the CF-EVRP in order to represent a just-in-time material handling system based on AGVs that takes into account time windows for delivery, limited operating range of the vehicles and distributed charging stations, as well as limitations in the road segments capacity. (ii) Design of problem instances to test the model's correctness and performance. (iii) Benchmark of a state-of-the-art SMT solver over the generated problem instances.

The paper is organized as follows: in Section 2 a formal description of the problem is provided, together with a mathematical model and detailed descriptions of its constraints. In Section 3, the results of the analysis over the set of generated problem instances is presented. Finally, conclusions are drawn in Section 4.

2 Problem Formulation

In the variations of the VRP mentioned in Section 1, the real-world map is abstracted and it is possible to travel directly from any customer to any other customer. For this reason it is possible to represent the problem as a complete undirected graph, where each customer is a node and two nodes are connected with each other by a weighted edge representing their distance. Customers are located at specific coordinates and the weight of the edge connecting two customers is their Euclidean or Manhattan distance.

However, in the CF-EVRP the capacity constraints on the road segments are taken into account and therefore, it is necessary to know where every vehicle is located at each time. Moreover, there can be several ways to travel from one customer to another, each with a different length. For these reasons

it is not possible to make an abstraction of the real world map. Instead, a weighted, directed graph represents the plant's actual layout; the nodes are the customers or the depot stations (as in the classical VRP), but they can also be intersections between road segments, while the edges represent the road segments and the edges' bi-dimensional weight represents their length and their capacity in terms of vehicles that can travel through it at the same time.

The following definitions are provided:

- Time Horizon: a fixed point in time in the future when all jobs will be assumed to end.

  $h = max_{n,n' \in N}(d_{nn'})$: the value of the longest road-segment

  B: an upper bound representing the time when all jobs should be finished

  $T = B + h$: the time horizon [1]

- Task: either a pickup or a delivery operation. A task is always associated with a node (see below) where the task is executed. Each and every task has a time window as an attribute, indicating the earliest and latest time at which a vehicle can execute the task. Unless explicitly stated, the time window for a task is the time horizon.

  $\mathcal{K}_j,\ \forall j \in \mathcal{J}$: the set of tasks of job j

  $L_{jk}, \in \mathcal{N}\ \forall j \in \mathcal{J}, k \in \mathcal{K}_j$: the location of task k of job j

  $\mathcal{P}_{jk} \subset \mathcal{K}_j,\ \forall j \in \mathcal{J}, k \in \mathcal{K}_j$: the set of tasks to execute before task k of job j

  $l_{jk},\ \forall j \in \mathcal{J}, k \in \mathcal{K}_j$: time window's lower bound for task k of job j

  $u_{jk},\ \forall j \in \mathcal{J}, k \in \mathcal{K}_j$: time window's upper bound for task k of job j

- Job: one or more pickup tasks and one delivery task. Pickup may have precedence constraints among them, while the delivery task for a job always happens after all pickups for that job are completed.

  $\mathcal{J}$: the set of jobs

- Vehicle: a transporter, e.g. an AGV, that is able to move between locations in the plant and perform pickup and delivery operations.

[1] The need to increase the value B is discussed on page C12.

$\mathcal{V}$: the set of all vehicles

$\mathcal{V}_j \subseteq \mathcal{V},\ \forall j \in \mathcal{J}$: set of vehicles eligible for job j

OR: the maximum operating range of the vehicles

C: the charging coefficient

D: the discharging coefficient

- Node: a location in the plant. A node can only accommodate one vehicle at the time unless otherwise specified.

  $\mathcal{N}$: the set of nodes

  $\mathcal{A}_n \subset N,\ \forall n \in N$: set of nodes with an incoming edge from node n

- Hub: a node that can accommodate an unbounded number of vehicles at the same time; nodes that are not hubs can only accommodate a single vehicle.

  $\mathcal{N}_H \subseteq N$: set of *hub* nodes

- Depot: a node at which one or more vehicles start and must return to after completing the assigned jobs. The depot can by definition accommodate an arbitrary number of vehicles at the same time, hence it is a hub node. Moreover, the depot is the only charging station of its vehicles.

  $\mathcal{F}$: the set of charging stations

  $s_i \in \mathcal{N},\ \forall i \in V$: starting node of vehicle i

- Edge: a road segment that connects two nodes.

  $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$: the set of edges

  $d_{nn'},\ \forall n, n' \in \mathcal{N}$: the length of the edge connecting nodes n and n' [2]

  $g_{nn'}, \forall n, n' \in \mathcal{N}$: the capacity of the edge connecting nodes n and n'

The requirements of the problem are summarized as follows:

- all jobs have to be completed; for a job to be completed a vehicle has to be assigned to it and visit the locations of the job's tasks within their respective time windows.

[2] It is assumed that one unit of distance is covered in one time-step, hence distance and duration are interchangeable

- when a vehicle is assigned to more than one job, it has to execute the job's tasks sequentially; all tasks of a job must be completed before it can execute any task of another job.
- there can be an arbitrary number of depots; vehicles have to return to the depot they were dispatched from.
- vehicles are powered by batteries with limited capacity but with the ability to recharge at a charging station. It is assumed that charging and discharging of the batteries is linear with respect to distance.
- different road segments in the plant have different capacities in terms of number of vehicles they can accommodate.
- pickup and delivery duration (the time when the pickup and delivery tasks respectively are executed) are considered to be zero, as these times can be considered negligible compared to the travelling time.
- not all vehicles are eligible to execute all jobs.

The set of variables used to build the model are:

x_{ij}: Boolean variables that evaluate to true if the i-th vehicle is assigned to the j-th job

y_{ijk}: Boolean variables that evaluate to true if the i-th vehicle is assigned to the k-th task of the j-th job

z_{ijkt}: Boolean variables that evaluate to true if the i-th vehicle serves the k-th task of the j-th job at time t

rc_{it}: Integer positive variables that keep track of the remaining operating range of vehicle i at time t

at_{int}: Boolean variables that evaluate to true if the i-th vehicle is at the n-th node at time t

$move_{int}$: Boolean variables that evaluate to true if the i-th vehicle decides to move to the n-th node at time t

The following logical operators are used to express cardinality constraints [17]:

AMO(a) : at most one variable of the set a evaluates to true;

AMN(a, n) : at most n variables of the set a evaluate to true;

If(c, o_1, o_2) : if condition c is *true* returns o_1, else returns o_2.

For further clarity, the constraint sets have been divided into sections, depending on the features of the system they model. The *job assignment* is concerned with the allocation of the vehicles to the jobs; the *vehicle movements* defines the rules for the vehicles to move through the plant; the *conflict-free routing* deals with the capacity constraints on the road segments; the *battery management* defines how the vehicles' batteries discharge and recharge according to their state.

2.1 Jobs Assignment

The following constraints are about assignment of jobs to vehicles, making sure that all jobs are assigned one and only one vehicle according to the specifications, and that such vehicle performs service to the job's tasks:

$$z_{ijkt} \implies at_{iL_{jk}t} \qquad \forall i \in \mathcal{V}, j \in \mathcal{J}, k \in \mathcal{K}_j, t = 0, \ldots, T \tag{C.1}$$

$$x_{ij} \implies \bigwedge_{\substack{i' \in \mathcal{V} \\ i \neq i'}} \neg x_{i'j} \qquad \forall i \in \mathcal{V}, j \in \mathcal{J} \tag{C.2}$$

$$y_{ijk} \implies \bigwedge_{\substack{i' \in \mathcal{V} \\ i \neq i'}} \neg y_{i'jk} \qquad \forall i \in \mathcal{V}, j \in \mathcal{J}, k \in \mathcal{K}_j \tag{C.3}$$

$$y_{ijk} \implies \bigvee_{t=0}^{T} z_{ijkt} \qquad \forall i \in \mathcal{V}, j \in \mathcal{J}, k \in \mathcal{K}_j \tag{C.4}$$

$$z_{ijk_1t} \implies \bigwedge_{\substack{k_2 \in \mathcal{P}_{jk_1} \\ t' = t, \ldots, T}} \neg z_{ijk_2t'} \qquad \forall i \in \mathcal{V}, j \in \mathcal{J}, k \in \mathcal{K}_j, t = 0, \ldots, T \tag{C.5}$$

$$x_{ij} \implies \bigwedge_{k \in \mathcal{K}_j} y_{ijk} \qquad \forall i \in \mathcal{V}, j \in \mathcal{J} \tag{C.6}$$

$$y_{ijk} \implies \bigvee_{t = l_{jk}, \ldots, u_{jk}} z_{ijkt} \quad \forall i \in \mathcal{V}, j \in \mathcal{J}, k \in \mathcal{K}_j \tag{C.7}$$

$$\neg x_{ij} \qquad \forall i \notin \mathcal{V}_j, j \in \mathcal{J} \tag{C.8}$$

(C.1) states that in order to serve a task at time t, a vehicle has to be at the task location at time t; (C.2) states that only one vehicle can be assigned to a job; similarly, (C.3) states that only one vehicle can be assigned to a task; (C.4) states that, in order to execute a task, a vehicle must be at the task location at some point in time; (C.5) enforces the precedence constraint among tasks of the same job; (C.6) states that, when assigned to a job, a vehicle must execute all tasks of the job; (C.7) states that a vehicle must execute tasks within their time windows; (C.8) states that only eligible vehicles can execute the corresponding job.

2.2 Vehicles' Movements

The following constraints are related to the vehicles' movements, specifying where vehicles can go given their current location, how long it will take to reach the next location, and how being at a location relates to performing a service at that location:

$$at_{is_i 0} \quad \forall i \in \mathcal{V} \tag{C.9}$$

$$at_{is_i B} \quad \forall i \in \mathcal{V} \tag{C.10}$$

$$\mathrm{AMO}_{n\in\mathcal{N}}(at_{int}) \quad \forall i \in \mathcal{V}, t = 0, \ldots, T \tag{C.11}$$

$$\mathrm{AMO}_{i\in\mathcal{V}}(at_{int}) \quad \forall n \in \mathcal{N} \setminus \mathcal{N}_H, t = 0, \ldots, T \tag{C.12}$$

$$\mathrm{AMO}_{n\in\mathcal{N}}(move_{int}) \quad \forall i \in \mathcal{V}, t = 0, \ldots, T \tag{C.13}$$

$$at_{int} \implies \neg move_{int} \quad \forall i \in \mathcal{V}, n \in \mathcal{N}, t = 0, \ldots, T \tag{C.14}$$

$$at_{int} \implies \bigwedge_{n' \notin \mathcal{A}_n} \neg move_{in't} \quad \forall i \in \mathcal{V}, n \in \mathcal{N}, t = 0, \ldots, T \tag{C.15}$$

$$(at_{int} \wedge \bigwedge_{n' \in \mathcal{N}} \neg move_{in't}) \implies at_{int+1} \quad \forall i \in \mathcal{V}, n \in \mathcal{N}, t = 0, \ldots, T-1 \tag{C.16}$$

$$(at_{int} \wedge move_{in't}) \implies \left(\bigwedge_{\substack{n'' \in \mathcal{N} \\ t' = t+1, \ldots, t+d_{nn'}-1}} \neg at_{in''t'} \wedge at_{in't+d_{nn'}} \right) \quad \forall i \in \mathcal{V}, (n, n') \in \mathcal{E}, t = 0, \ldots, T - d_{nn'} \tag{C.17}$$

$$(z_{ij_1k_1t_1} \wedge z_{ij_1k_2t_2}) \implies \bigwedge_{\substack{j_2 \in J, j_1 \neq j_2 \\ k' \in K_{j_2} \\ t' = t_1+1, \ldots, t_2}} \neg z_{ij_2k't'} \quad \forall i \in \mathcal{V}, j_2 \in \mathcal{J}, k_1, k_2 \in \mathcal{K}_{j_1}, k_1 \neq k_2, t_1 = 0, \ldots, T, t_2 = t_1, \ldots, T \tag{C.18}$$

(C.9) states that all vehicles are at their respective depots at time zero; (C.10) states that all vehicles are at the depot at time B, these constraints force the vehicles to return to the depot after they execute the jobs they are assigned to; (C.11) states that a vehicle can be at most in one location at a time.

Normally, it would make sense to state that a vehicle should be *exactly* in one place at a time. However, given the way the vehicle movement is modeled, this constraint needs to be relaxed because moving can take more than one time-step; (C.13) states that a vehicle can move to at most one location at a time; (C.14) forbids a vehicle to move to the node where it already is, and (C.15) states that if vehicle i at time-step t is at node n, it cannot move to any non-adjacent node; (C.12) states that at most one vehicle can, at the same time-step, be at a node that is not a *hub*; (C.16) states that, if a vehicle is not moving at a certain time-step, it will be at the same location at the next time-step; (C.17) states that, if a vehicle moves to a new location, it will be at no node while it is transiting on the edge connecting the start and arrival nodes, and it will be at the arrival node after as many steps as it takes to traverse the edge, defined by the edge's weight. It is for this reason that constraint (C.11) needs to be relaxed. Also, in (C.17) $t' = t + 1, \ldots, t + d_{nn'} - 1$, therefore $t = 0, \ldots, T - d_{nn'}$. This means that there are no constraints applying to events taking place in the interval $[T - d_{nn'}, T]$. For this reason it is necessary to extend the time horizon to $T = B + h$, the steps $B, \ldots, h$ are not actually used in the model but this way it is guaranteed that all constraints apply for $T = 0, \ldots, B$; (C.18) states that if a vehicle is executing two tasks k_1 and k_2 of job j, then the vehicle cannot execute a task of any other job different from j in between. This constraints is used to make sure that if a vehicle is selected for multiple jobs, it has to execute them in sequence (i.e., finish all tasks of a job before executing any other task).

2.3 Conflict-Free Routing

The following constraints are related to the conflict-free routing, specifying how one or more vehicle's location affects the other vehicles' movements:

$$\bigwedge_{i\in\bar{\mathcal{V}}} at_{int} \implies \bigwedge_{n'\in\mathcal{A}_n} \mathrm{AMN}_{i\in\bar{\mathcal{V}}}(move_{in't}, g_{nn'})$$
$$\{\bar{\mathcal{V}} \in 2^{\mathcal{V}} \mid |\bar{\mathcal{V}}| > g_{nn'}\}, n \in \mathcal{N}, t = 0, \ldots, T \tag{C.19}$$

$$\bigwedge_{i_1\in\bar{\mathcal{V}}_1} at_{i_1nt} \wedge \bigwedge_{i_2\in\bar{\mathcal{V}}_2} at_{i_2n't} \wedge \bigwedge_{i_1\in\bar{\mathcal{V}}_1} move_{i_1n't} \implies$$
$$\bigwedge_{t'=t}^{t+d_{nn'}} \mathrm{AMN}_{i_2\in\bar{\mathcal{V}}_2}\big(move_{i_2nt'}, g_{nn'} - |\bar{\mathcal{V}}_1|\big)$$
$$\{\bar{\mathcal{V}}_1 \in 2^{\mathcal{V}} \mid |\bar{\mathcal{V}}_1| \leq g_{nn'}\}, t = 0, \ldots, T - d_{nn'}$$
$$\{\bar{\mathcal{V}}_2 \in 2^{\mathcal{V}} \mid |\bar{\mathcal{V}}_2| \geq g_{nn'} - |\bar{\mathcal{V}}_1|\}, (n, n') \in \mathcal{E} \tag{C.20}$$

(C.19) states that if n vehicles are on the same node, they cannot transit on the same edge having capacity $n-1$ at the same time-step; (C.20) states that if n vehicles are on two adjacent nodes and decide to traverse the edge connecting them and whose capacity is $n+m$, at most m vehicles can traverse the edge from the other node before the n vehicles are done transiting.

2.4 Battery Management

The following constraints are related to the battery management, i.e., where the vehicles can charge, how fast, and how far they can travel before needing to recharge:

$$rc_{it} \geq 0 \wedge rc_{it} \leq OR \qquad \forall i \in \mathcal{V}, t = 0, \ldots, T \tag{C.21}$$

$$(at_{int} \wedge move_{in't}) \implies \bigwedge_{t'=t+1,\ldots,t+d_{nn'}+1} rc_{it'} = rc_{it'-1} - D$$

$$\forall i \in \mathcal{V}, (n, n') \in \mathcal{E}, t = 0, \ldots, T - d_{nn'} \tag{C.22}$$

$$(at_{int} \wedge \bigwedge_{n' \in \mathcal{N}} \neg move_{in't}) \implies rc_{it+1} = rc_{it}$$

$$\forall i \in \mathcal{V}, n \in \mathcal{N} \setminus \mathcal{F}, t = 0, \ldots, T - 1 \tag{C.23}$$

$$(at_{int} \wedge \bigwedge_{n' \in \mathcal{N}} \neg move_{in't}) \implies rc_{it+1} = rc_{it} + C$$

$$\forall i \in \mathcal{V}, n \in \mathcal{F}, t = 0, \ldots, T - 1 \tag{C.24}$$

(C.21) defines the domain of the rc_{it} variables; (C.22) states that, if a vehicle is moving, its energy decreases at each time-step according to the discharge coefficient; (C.23) states that if a vehicle is not moving, its energy level does not change. It is necessary to specify that the vehicle is at some node because if the vehicle is at no node, it is moving; (C.24) states that if a vehicle is at a charging station, its energy level increases at each time-step according to the charging coefficient.

2.5 Objective Function

Finally, the cost function to minimize (C.25) is the total travelled distance, calculated as the cumulative distance travelled by each and every vehicle:

$$\sum \mathrm{If}((at_{int} \wedge move_{in't}), d_{nn'}, 0)$$

$$\forall i \in \mathcal{V}, n \in \mathcal{N}, t = 0, \ldots, T - 1 \tag{C.25}$$

3 Computational Analysis

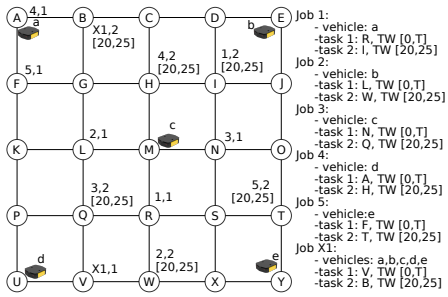

Figure 1: Test Case with five vehicles *a* to *e* that have to execute six jobs *1* to *5* and *X1*, each consisting of a pickup with time window [0,T] and a delivery with time window [20,25].

In order to evaluate the model presented in Section 2 we generated a set of problem instances, defined by a weighted, directed graph that represents a hypothetical plant layout; The number of nodes is in this case a parameter that affects the problem size, as well as the number of edges. In this work we restricted the capacity of all road-segments to be *one*. For the graph it is specified which nodes are *hubs*. Another parameter of the instance is the number of vehicles available; to each vehicle a starting node is assigned; more vehicles can start from the same node (in this case the node must be a *hub*). A list of jobs to execute is given; each job is characterized by a number of tasks and the vehicle(s) eligible to execute it. In turn, each task is characterized by a location (node), a precedence list, and a time window. Other parameters of the instance are the charging and discharging coefficients, as well as the operating range of the vehicles and the time horizon, by when all jobs should be finished and the vehicles should be back at their starting locations.

Figure 1 shows what a problem instance may look like; there are five depot stations located at nodes A, E, M, U, and Y, which are also the charging stations in the problem. There is no *hub*. Vehicles are initially located as indicated in the figure and they have to return to their initial location before the last time-step. All edges in this problem have weight *one*, the charging and discharging coefficients are both *one*, the time horizon T is equal to *35*, and the vehicles operating range is *15*. Tasks are reported in the figure next to their location; for instance, *task 2* of *job 1* is at node *I*. Time windows,

Table 1: Evaluation of the CF-EVRP over a set of generated problem instances. Instances are sorted by the parameters N-V-J (nodes, vehicles, jobs), value of edge reduction, and time horizon. For each resulting class, the number of instances solved within the time limit of 10800 seconds (Sol) is reported, as well as the average generation time (GT, in seconds), and the amount (SC) and average solving time (ST, also in seconds) for the instances that turned out to be satisfiable (on the left), and unsatisfiable (on the right). The symbol "-" means that no instance for that category was either satisfiable or unsatisfiable, depending on where the symbol appears.

Time Horizon		N-V-J															
		15-3-5 Edge Reduction						25-4-7 Edge Reduction						35-6-8 Edge Reduction			
		0		25		50		0		25		50		0		25		50	
15	Sol	5/5		5/5		5/5		5/5		5/5		5/5		5/5		5/5		5/5	
	GT	12		14		17		80		96		59		87		197		137	
	ST	2	3	3	3	-	5	-	34	-	35	-	15	-	33	-	108	-	218
	SC	3	2	2	3	0	5	0	5	0	5	0	5	0	5	0	5	0	5
20	Sol	5/5		5/5		5/5		5/5		5/5		5/5		5/5		5/5		5/5	
	GT	22		31		34		117		148		133		149		241		246	
	ST	12	2	22	4	-	169	281	34	1517	107	-	1669	1619	37	2375	144	-	818
	SC	4	1	4	1	0	5	3	2	2	3	0	5	2	3	1	4	0	5
25	Sol	5/5		5/5		4/5		5/5		4/5		2/5		2/5		4/5		5/5	
	GT	38		47		55		192		222		233		257		295		391	
	ST	19	29	69	61	204	39	1380	292	3327	246	1046	100	1911	-	2801	2551	7754	2826
	SC	4	1	4	1	2	2	4	1	3	1	1	1	2	0	1	3	1	4
30	Sol	5/5		5/5		3/5		4/5		3/5		2/5		1/5		2/5		2/5	
	GT	64		71		97		371		339		414		568		591		607	
	ST	557	-	549	-	987	1152	5995	-	3916	-	9325	3414	6052	-	9812	703	9503	629
	SC	5	0	5	0	2	1	4	0	3	0	1	1	1	0	1	1	1	1

if they exist, are reported below the task name, in between square brackets; for instance, the time window for *task 1* of *job 2* is *[20,25]*. In this problem jobs *1* to *5* can only be executed by vehicles *a* to *e* respectively, while *Job x1* can be executed by any vehicle. For this particular instance the solver took about 90 seconds to generate the model and 540 seconds to find the first feasible solution, which correspond to a total travelled distance of *102*. Jobs *1* to *5* are executed by vehicles *a* to *e* respectively, while *Job x1* is executed by *Vehicle d* after executing *Job 4* and recharging at *Node A*.

In the computational analysis we evaluated 160 problem instances structured as explained in the previous paragraphs. The parameters we used for the problems generation are:

- *N-V-J* (nodes, vehicles, and jobs, respectively). *N-V-J* can be either *15-3-5*, or *25-4-7*, or *35-6-8*. The corresponding layouts are grids of size 5×3, 5×5 (as in Figure 1), and 5×7. Each job is composed by one pickup and one delivery;
- *Edge Reduction* can be either *0*, or *25* or *50*. The value *0* correspond to a complete grid, as in Figure 1; the value *25* corresponds to a grid where 3, 6, and 9 pairs of edges (i.e. (A, B) and (B, A)) are removed for N equal to 15, 25, and 35 respectively; the value *50* corresponds to a grid where 6, 12, and 18 pairs of edges are removed for N equal to 15, 25, and 35 respectively.
- Time Horizon can be either *15*, *20*, *25*, or *30*.
- All edges have weight equal to *one*.

For each combination of the above-mentioned parameters, five different problems are generated by randomly assigning the starting node for each type of vehicle, the operating range, the charging coefficient, the number of vehicles available per each type, the type of vehicle required for each job, and the location of the jobs' tasks.

Problem instances are designed to be on average satisfiable if the capacity constraints are not included. This can be achieved by acting on the operating range of the vehicles and on the time windows of the jobs. The goal is to produce both satisfiable and unsatisfiable problem instances in order to evaluate the performance of the solver in both cases. Moreover, unsatisfiability can be caused by different factors, such as too tight time windows and/or time

horizon, not sufficient operating range, or too low connectivity of the graph (in this case there are not enough paths to reach the desired locations without getting into another vehicle's way). By tuning the previously defined parameters it is possible to trigger different type of unsatisfiability and evaluate the solver performance accordingly.

For the analysis we used Z3 4.8.9. The time limit is set to 10800 seconds (three hours); the model generation time is measured separately, since it is implementation-dependent and can be dealt with using more efficient formulations, as discussed in our previous work [18]. All the experiments were performed on an *Intel Core i7 6700K, 4.0 GHZ, 32GB RAM* running *Ubuntu-18.04 LTS*. Though Z3 allows for optimization of the objective function, the size of the problems evaluated is such that no optimum is expected to be found in any reasonable time. Therefore Z3 is set to find satisfiable (hence sub-optimal) solutions [3].

In Table 1 the results of the computational analysis are summarized. For each combination of the values introduced in the previous paragraph, it is reported the number of solved problems (out of *five*), the average model generation time, and the average solving time, both for the instances that turned out to be satisfiable and for the ones that did not; the amount of satisfied and unsatisfied instances is also reported. As expected, all the parameters are directly proportional to the model generation time, since the model size increases as they increase. The solving time is also directly affected by the increase of the parameters' value. In general, given problems of the same class, proving unsatisfiability seems to be faster than finding a feasible solution; there are exceptions, especially for values of edge reduction of 25 and 50. Even when the problem is satisfiable, finding a solution is harder for a smaller number of edges, since the number of possible conflict-free routes decreases. For *N-V-J* equal to 25-4-7 and 36-6-8 and a time horizon of 15 time units all problems turn out to be unsatisfiable; this result does not come unexpected, since the graph and the gap between number of vehicles and number of jobs are too large to execute all jobs and go back to the starting point in time. We assumed that an obvious unsatisfiability could be faster to prove, but the results seem to contradict our hypothesis.

[3] The implementation of the model presented in Section 2, as well as the problem instances are available at `https://github.com/sabinoroselli/VRP.git`.

4 Conclusion

In this paper we formulated a model to represent just-in-time material handling systems based on AGVs that takes into account limited operating range and road segments capacity. We also designed a set of problem instances to test the model, implemented for the state-of-the-art SMT solver Z3. The solver turned out to be able to handle medium and large models; however, given their complexity, these models correspond to relatively small systems. The approach presented can nonetheless be used to solve simplified problems to provide initial feasible solutions or as a part of a compositional algorithm. Also, it can be used to test the correctness of alternative approaches because, given enough running time, it can provide the true optimum. Finally, the benchmark set we generated can be used to compare performance of different approaches.

References

[1] K. Azadeh, M. deKoster, and D. Roy, "Robotized warehouse systems: Developments and research opportunities," *ERIM Report Series Research in Management*, no. ERS-2017-009-LIS, 2017.

[2] K.-h. Lai and T. E. Cheng, *Just-in-time logistics*. CRC Press, 2016.

[3] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse, "The vehicle routing problem: State of the art classification and review," *Computers & Industrial Engineering*, vol. 99, pp. 300–313, 2016.

[4] M. Desrochers, J. Desrosiers, and M. Solomon, "A new optimization algorithm for the vehicle routing problem with time windows," *Operations research*, vol. 40, no. 2, pp. 342–354, 1992.

[5] M. Schneider, A. Stenger, and D. Goeke, "The electric vehicle-routing problem with time windows and recharging stations," *Transportation Science*, vol. 48, no. 4, pp. 500–520, 2014.

[6] N. N. Krishnamurthy, R. Batta, and M. H. Karwan, "Developing conflict-free routes for automated guided vehicles," *Operations Research*, vol. 41, no. 6, pp. 1077–1090, 1993.

[7] E. Thanos, T. Wauters, and G. Vanden Berghe, "Dispatch and conflict-free routing of capacitated vehicles with storage stack allocation," *Journal of the Operational Research Society*, pp. 1–14, 2019.

[8] K. Murakami, "Time-space network model and MILP formulation of the conflict-free routing problem of a capacitated AGV system," *Computers & Industrial Engineering*, 2020.

[9] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, part I: Route construction and local search algorithms," *Transportation science*, vol. 39, no. 1, pp. 104–118, 2005.

[10] S. Riazi, T. Diding, P. Falkman, K. Bengtsson, and B. Lennartson, "Scheduling and routing of agvs for large-scale flexible manufacturing systems," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2019, pp. 891–896.

[11] B. M. Baker and M. Ayechew, "A genetic algorithm for the vehicle routing problem," *Computers & Operations Research*, vol. 30, no. 5, pp. 787–800, 2003.

[12] Y.-J. Gong, J. Zhang, O. Liu, R.-Z. Huang, H. S.-H. Chung, and Y.-H. Shi, "Optimizing the vehicle routing problem with time windows: A discrete particle swarm optimization approach," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 2, pp. 254–267, 2011.

[13] S. F. Roselli, K. Bengtsson, and K. Åkesson, "SMT solvers for job-shop scheduling problems: Models comparison and performance evaluation," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2018, pp. 547–552.

[14] ——, "SMT solvers for flexible job-shop scheduling problems: A computational analysis," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2019.

[15] C. Barrett and C. Tinelli, "Satisfiability modulo theories," *Handbook of model checking*, pp. 305–343, 2018.

[16] L. De Moura and N. Bjørner, "Satisfiability modulo theories: Introduction and applications," *Commun. ACM*, vol. 54, no. 9, pp. 69–77, Sep. 2011, ISSN: 0001-0782.

[17] C. Sinz, "Towards an optimal CNF encoding of boolean cardinality constraints," in *International conference on principles and practice of constraint programming*, Springer, 2005, pp. 827–831.

[18] S. F. Roselli, K. Bengtsson, and K. Åkesson, "Compact representation of time-index job shop problems using a bit-vector formulation," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2020, pp. 1590–1595.

PAPER D

A Compositional Algorithm for the Conflict-Free Electric Vehicle Routing Problem

Sabino Francesco Roselli, Per-Lage Götvall, Martin Fabian, Knut Åkesson

2022 IEEE Transactions on Automation Science and Engineering (TASE)
pp. 1405 - 1421, May. 2022.
 DOI: 10.1109/TASE.2022.3169949

The layout has been revised.

Abstract

The Conflict-Free Electric Vehicle Routing Problem (CF-EVRP) is an extension of the Vehicle Routing Problem (VRP), a combinatorial optimization problem of designing routes for vehicles to visit customers such that a cost function, typically the number of vehicles or the total travelled distance, is minimized. The problem finds many logistics applications, particularly for highly automated logistic systems for material handling. The CF-EVRP involves constraints such as time windows on the delivery to the customers, limited operating range of the vehicles, and limited capacity on the number of vehicles that a road segment can accommodate at the same time. In this paper, the compositional algorithm ComSat for solving the CF-EVRP is presented. The algorithm iterates through the sub-problems until a globally feasible solution is found. The proposed algorithm is implemented using an optimizing SMT-solver and is evaluated against an implementation of a previously presented monolithic model. The soundness and completeness of the algorithm are proven, and it is benchmarked on a set of generated problems and found to be able to solve problems of industrial size.

Note to Practitioners

The need to define and solve the CF-EVRP relates to an industrial application where a fleet of autonomous robots navigate in a heterogeneous environment, shared with humans and other vehicles and obstacles. To allow for a low-level trajectory controller to handle dynamic obstacles, like humans and fork-lifts, the CF-EVRP includes capacity constraints on the road segments. This increases the problem complexity, and thus requires to trade off optimality for feasability; this so to get solutions in reasonable time with respect to how long ahead the jobs to schedule are known. The overall problem is to find feasible solutions that satisfy all constraints while avoiding travelling unnecessarily long routes, and at the same time meet the stipulated time-windows to deliver material just-in-time. The compositional algorithm (ComSat) presented

in this work is based on the idea to break down the overall scheduling problem into sub-problems that are easier to solve, and then to build a schedule based on the solutions of the sub-problems. ComSat is designed to work well for industrial scenarios where there are good reasons to believe that feasible solutions do exist. This seems a reasonable assumption as in an industrial setting a sufficient number of mobile robots can typically be assumed to be available.

1 Introduction

The use of mobile robots for *just-in-time* deliveries is of considerable interest for modern manufacturing facilities [1]. The problem treated in this paper originates from an industrial need to use a fleet of Automated Guided Vehicles (AGVs) that run through the plant to deliver components to workstations just-in-time for them to be used. In this scenario, the scheduler needs to consider several types of constraints in addition to the time constraints. (i) AGVs have a limited operating range and need to recharge their battery when the state-of-charge becomes low. (ii) Jobs have specific requirements on the AGV to execute them where only some AGVs can handle some jobs. (iii) The plant layout may limit the AGV's freedom of movement; for instance, a passage may not be large enough to accommodate more than a fixed number of AGVs at a time. Thus, we define the *capacity* of the road segments, intersections, and workstations and include *capacity constraints* in the problem formulation. A schedule is *conflict-free* if it fulfills the capacity constraints.

The constraints discussed above increase the complexity of the problem, and even finding feasible solutions can take a long time. A solution to a standard Vehicle Routing Problem (VRP) can be computed within minutes for up to 100 customers and just as many vehicles [2]. On the other hand, when charging times, capacity constraints, and multiple assignments of routes to vehicles come into play, a problem involving 10 vehicles and 20 customers can be considered large, and may take hours to find a feasible solution for.

In industrial applications there often occur changes that require re-scheduling, and a new schedule must be found quickly, else it may be outdated once computed. At the same time, production cannot be put on hold while a new schedule is computed. For these reasons, the compositional algorithm presented in this work aims to find feasible solutions quickly, rather than optimal

ones.

There exist both approximate and exact algorithms to solve the VRP. For relatively small-size problem instances, mixed-integer linear programming (MILP) [3] solvers can find feasible or even optimal solutions in reasonable time. However, standard MILP-solving techniques do not scale well, so specific-purpose algorithms involving local search [4], Benders decomposition [5], or stochastic methods [6], [7] have been proposed. Recent work focusing on fleets of electric vehicles [8], as well as conflict-free routing [9] show applications of such approaches to real-world problems.

In [10] a comparison of using MILP as an exact method, and a set-based particle-swarm optimization algorithm as an approximate method, is made for solving the VRP with time windows (VRPTW [11]). The comparison shows that neither method dominates the other in terms of running time and quality of the solutions. On the other hand, the advantage of using general-purpose MILP-solvers is that additional constraints can be easily added to handle extensions of the original problem. At the same time this may be non-trivial, if at all possible, for a tailor-made algorithm.

The specific scheduling and routing problem treated in this paper, called the *Conflict-Free Electric Vehicle Routing Problem* (CF-EVRP), does involve additional constraints such as limited operating range of the robots, and capacity constraints, thus a general-purpose solver is used. In [12], [13] optimizing Satisfiability Modulo Theory (SMT [14], [15]) solvers outperformed MILP solvers on combinatorial scheduling problems such as Job Shop Problems (JSP) involving many logical constraints. The natural abilities of SMT solvers to natively handle combinatorial constraints make them well suited to handle CF-EVRP.

A monolithic model of the CF-EVRP is presented in [16] and solved using the SMT-solver Z3 [17]. Already relatively small problems, with only a few vehicles and jobs, result in hundreds of thousands of variables and constraints due to the discretization of time used to model capacity constraints.

In order to avoid time discretization and be able to quickly find feasible solutions to large problems of the CF-EVRP, [18] presents a compositional algorithm for a slightly restricted CF-EVRP. This algorithm scales better than the monolithic one. However, it does not account for typical industrial features such as service time and multiple depots.

This work introduces *ComSat* (Compositional Satisfiability), an extension

of the compositional algorithm presented in [18]. The contributions of this paper are: (i) a description of CF-EVRP, a problem that tackles both the issue of limited operating range of the vehicles, and the limited capacity of the segments in a road network; the CF-EVRP from [18] is also generalized so that vehicles can be located at multiple depots, and service times at the customers are accounted for; (ii) a decomposed mathematical formulation of CF-EVRP; (iii) the ComSat algorithm designed for the decomposed problem formulation; (iv) a proof of ComSat's soundness and completeness under given restrictions; (v) an evaluation of ComSat on a set of generated instances of the CF-EVRP.

The paper is organized as follows. Section 2 introduces previous works on the topic and puts this work in context. Section 3 provides a formal description of the problem. In Section 4, ComSat is introduced. Proof of soundness and completeness is given in Section 5. In Section 6, the results of the analysis over a set of problem instances are presented. Finally, conclusions are drawn in Section 7.

2 Literature Review

The VRP [19] is a classical problem, formulated by Dantzig and Ramser, that searches for optimal routes for a fleet of robots to visit a set of customers. A large number of studies have introduced variations on the original problem, as well as techniques to solve them. The vehicle routing problem with time-windows (VRPTW) is an extension of the VRP where customers have to be visited within given time windows [11]. A related problem is discussed in [20] where the routes to visit customers are dynamically designed based on the current state of the other vehicles. Another variation of the problem involves the possibility of Multiple Depots (MDVRP). In [21], the MDVRP is decomposed into assignment of vehicles to customers, and design of routes for vehicles to visit their assigned customers. In [22], the MDVRP is solved by means of genetic algorithms. In [23] the problem of limited capacity of the road segments is tackled and conflict-free routes for AGVs are computed by means of column generation. The work in [24] presents one of the first and most relevant works involving conflict-free routing in combination with scheduling of jobs for flexible manufacturing systems. Similarly to [21], the authors break down the problem into a scheduling problem, solved by constraint programming, where

vehicles are assigned to jobs, and a routing problem, solved by MILP, where routes for the vehicles are designed. We took inspiration from this approach and further broke down the problem into more sub-problems to be able to handle the different constraints, in particular the limited operating range of the vehicles.

For electric vehicles, model formulations need to take into account the vehicle's limited operating range and non-negligible charging time as well. In [25] a branch and cut algorithm to solve a VRP with satellite facilities is presented, vehicles can stop to replenish their cargo and continue delivering goods until the end of their shift. Satellite facilities are also treated by [26], that models these intermediate points as charging stations and solves the problem by means of a combination of neighborhood and tabu search.

Autonomous vehicles are increasingly used to deliver material across manufacturing plants. In [27], AGVs are scheduled for jobs and routed through a plant by means of Petri net decomposition. In [28], a hybrid evolutionary algorithm is implemented to solve a multi-objective AGV scheduling problem in a flexible manufacturing system. In this work, the authors consider the vehicles' battery charge, but do not take into account road segments' capacity. In [29], a multi-objective AGV scheduling problem is solved by means of adaptive-genetic algorithms. Unlike standard genetic algorithms, these adjust the hyperparameters, improving convergence accuracy and speed. The authors consider a plant with a grid-like road network, but road segment capacities are not considered. In [30] an integrated approach to deal with line balancing and material handling by means of AGVs is presented. A stochastic algorithm is used to assign jobs to the workstations and AGVs are scheduled to deliver the components to execute the jobs. However, road segment capacity constraints are not considered. In [31], a matheuristic (a combination of metaheuristics and mathematical programming) to schedule a heterogeneous fleet of AGVs is presented, having different travel speed and cost, charging/discharging rate, and capability to serve different customers. But again, road capacity constraints are not treated.

As AGVs are used in manufacturing plants with limited capacity of the road segments, a growing attention has been paid to the problem of designing conflict-free routes. In [32], an ant colony algorithm is applied to the problem of job shop scheduling and conflict free routing of AGVs. While road segment capacity constraints are considered the limited operating range of the vehicles

is not considered. In [33], a collision-free path planning for multi AGV systems based on the A^* algorithm is presented. In this work, the environment is modeled as a grid, and conflicts can originate from vehicles occupying the same spot on the grid at the same time; the vehicles' operating range and ability to recharge is not considered. In [9], a heuristic approach to solve the conflict-free routing problem with storage allocation is presented; in this work limited operating range and battery charge are not considered. In [34], a MILP formulation to design conflict-free routes for capacitated vehicles is presented. This is an exact method, but it can only solve relatively small problem instances. In [35] a hybrid evolutionary algorithm to deal with conflict-free AGV scheduling in automated container terminals is presented. In this work, only a limited portion of the map is prone to conflicts, with all road segments allowing to travel in both directions simultaneously; also, charging of the vehicles' batteries is not considered. In [36] the authors present a new model formulation for the VRPTW that restricts the problem to only difference logic (a fragment of linear arithmetic) constraints, in order to exploit the strength of Z3 in dealing with this particular fragment.

From the literature review it emerges that over the last two decades there has been a growing number of studies dedicated to AGV-based material handling systems. There is usually a large overlap of features tackled in each work, and approximate methods are likely to be used to solve large problem instances, due the problem being too complex to be solved by exact algorithms. We apply a graph-based concept similar to [33], with nodes and edges of the graph representing the plant. Similarly to [36] we do exploit the strength of Z3 in dealing with difference logic by turning the *Assignment Problem* and the *Capacity Verification Problem* (see Section 4) into JSPs, that can be described using difference logic. However, to the best of our knowledge, there is no work that tackles within the same formulation both the limited operating range, with the necessity to recharge the vehicles' batteries, and the limited capacity of the road segments, requiring to schedule conflict-free routes. Industrially, both limitations are required to be included. The CF-EVRP includes all these limitations and ComSat exploits the decomposed formulation of the CF-EVRP to find feasible solutions.

3 Problem Definition and Notation

In the CF-EVRP the plant layout is represented by a finite, strongly connected, weighted, directed graph, where edges represent road segments and nodes represent either intersections between road segments or locations of customers. A customer is defined by a unique (numerical) identifier, a location, and a time window, i.e., a lower and an upper bound that represent the earliest and latest arrival time allowed to serve the customer. The word *customer* is typically used to denote a location where a vehicle is to pick up or drop off goods. The word *task* will be used synonymously. Edges have two attributes representing a road segment's length and its capacity in terms of number of vehicles that it can simultaneously accommodate.

The following definitions are provided:

- Node: a location in the plant. A node can only accommodate one vehicle at a time unless it is a *hub* node.

  $\mathcal{N}$: a finite set of nodes.

  $\mathcal{N}H \subseteq N$: the set of hub nodes, nodes that can accommodate an arbitrary number of vehicles.

- Edge: a road segment that connects two nodes.

  $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$: the finite set of edges.

  $\bar{e},\ \forall e \in \mathcal{E}$: the reverse edge of an edge e.

  $d_{nn'} \in \mathbb{R},\ \forall \langle n, n' \rangle \in \mathcal{E}$: the length of the edge connecting nodes n and n'.

  $g_{nn'} \in \{1, 2\},\ \forall \langle n, n' \rangle \in \mathcal{E}$: the capacity of the edge connecting nodes n and n'

- Time horizon: a fixed point of time in the future when all jobs are assumed to have ended.

  T: the time horizon.

- Job: a set of tasks that must be executed within the same route and without executing any task belonging to a different job in between. Typically, a job is the pickup of parts from the warehouse and the delivery to the due workstation; however, in general a job can have an arbitrary

number of tasks. No task belonging to another job should be executed in the same route until all tasks belonging to the current job are completed.

$\mathcal{J}$: the finite set of jobs.

- Task: either a pickup or a delivery operation (there is no need to distinguish between them, as both are modelled in the same way). A task is always associated with a node where it is executed, and has a time window indicating the earliest and latest time at which it can be executed. Unless explicitly given, the time window spans the entire time horizon $[0, T]$. Also, each task is associated with a precedence list that states what tasks have to be executed before it. This list may include any other task in the problem definition. Finally, for each task a service time is defined.

  $\mathcal{K}$ the finite set of all tasks.

  $\mathcal{K}_j \subseteq \mathcal{K},\ \forall j \in \mathcal{J}$: the finite set of tasks of job j. (note that the tasks set is partitioned into subsets based on the jobs, i.e., $\mathcal{K}_i \cap \mathcal{K}_j = \varnothing\ \forall i \neq j,\ i, j \in \mathcal{J}$)

  $L_k \in \mathcal{N},\ \forall j \in \mathcal{J},\ k \in \mathcal{K}_j$: the location of task k.

  $\mathcal{P}_k \subset \mathcal{K}_j,\ \forall j \in \mathcal{J},\ k \in \mathcal{K}_j$: the set of tasks to execute before task k.

  $l_k \in \mathbb{R},\ \forall j \in \mathcal{J},\ k \in \mathcal{K}_j$: the time window's lower bound for task k.

  $u_k \in \mathbb{R},\ \forall j \in \mathcal{J},\ k \in \mathcal{K}_j$: the time window's upper bound for task k.

  $S_k \in \mathbb{R},\ \forall j \in \mathcal{J},\ k \in \mathcal{K}_j$: the service time of task k.

- Depots: nodes at which one or more vehicles start and must return to after completing the assigned jobs. A depot can accommodate an arbitrary number of vehicles at the same time, thus all depots are hubs.

  $\varnothing \subset \mathcal{O} \subseteq \mathcal{N}H$: the set of depots.

  $\mathcal{S} = \{\, s_o \,|\, o \in \mathcal{O} \,\}$: the set of dummy tasks representing the start from depot o.

  $\mathcal{F} = \{\, f_o \,|\, o \in \mathcal{O} \,\}$: the set of dummy tasks representing the arrival at depot o.

  The sets $\mathcal{S}$ and $\mathcal{F}$ are disjoint with each other and with all task sets $\mathcal{K}_j,\ \forall j \in \mathcal{J}$.

- Vehicle: a transporter, e.g. a mobile robot, that is able to move between locations in the plant and perform pickup and delivery operations.

  $\mathcal{V}$: the finite set of all vehicles.

  $\mathcal{V}_j \subseteq \mathcal{V},\ \forall j \in \mathcal{J}$: set of vehicles eligible for job j.

  $OR \in \mathbb{R}^+$: the vehicles' maximum operating range (constant).

  $C \in \mathbb{R}^+$: the charging coefficient (constant).

  $D \in \mathbb{R}^+$: the discharging coefficient (constant).

  $\rho \in \mathbb{R}^+$: a coefficient to scale remaining charge into remaining operating range (constant).

  $v \in \mathbb{R}^+$: vehicle speed (constant) while moving.

The requirements of the problem are:

- All jobs have to be completed; for a job to be completed a vehicle has to be assigned to it and visit the locations of the job's tasks according to the tasks' sequence and within their respective time windows.

- Vehicles are not allowed to arrive at the task's location before the time window's lower bound and wait there (many other VRP formulations, allow such waiting).

- Vehicles are powered by batteries with limited capacity but with the ability to recharge at the depots. It is assumed that state of charge increases proportionally to the time spent at the depot and decreases proportionally to the travelled distance. Also, vehicles travel at constant speed v or they are stationary.

- Multiple depots; vehicles have to return to the depot they were dispatched from and can only recharge their batteries there (without queuing).

- A non-empty subset of vehicles is eligible for each job.

- All vehicles have the same operating range and start at full charge; whenever they return to the depot they charge to full before becoming available again;

- Two additional jobs are added for each depot: *start* and *end*; they are needed in the *Routing Problem* to make sure that routes begin and end at the depot. Both *start* and *end* have only one task located at the depot they represent, service time equal to zero, and the entire time horizon as time window.

- Road segment capacities constrain the number of vehicles a road segment can simultaneously accommodate.

- Only (non-cyclic) *paths*, that is, finite sequences of edges that join sequences of distinct vertices, are considered.

3.1 Example of the CF-EVRP

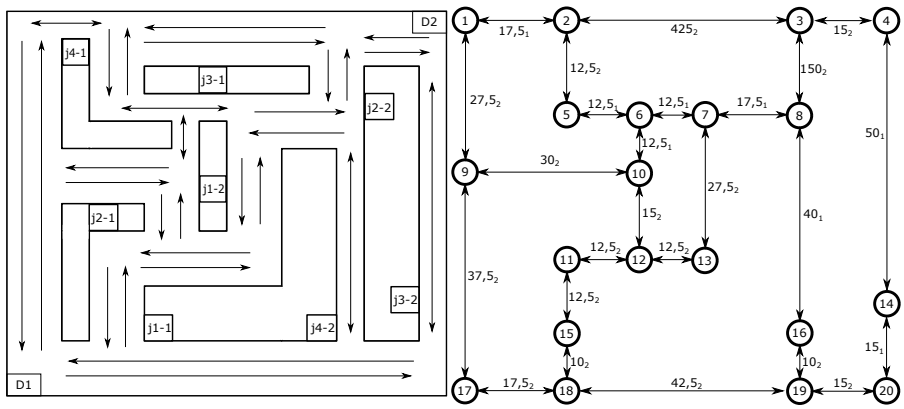

Figure 1: Problem instance of the CF-EVRP picturing a hypothetical plant (left) where two depots ($D1$, $D2$) accommodate four vehicles ($v1$, $v2$, and $v3$, $v4$ respectively), available to execute four jobs ($j1$, $j2$, $j3$, $j4$), each composed by two tasks (1, 2). The plant road segments are abstracted into a strongly connected, directed, weighed graph (right).

Fig. 1 shows an example of the CF-EVRP, where four AGVs are available to execute four jobs, each composed by two tasks (the squares distributed over the plant). Each task is marked by a numeric code where the first digit refers to the job and the second digit indicates the task number. On the right is shown how the plant layout is abstracted into a strongly connected, directed,

weighted graph (more on this below). The nodes represent the intersections of road segments in the plant; if a task's location is close enough to an intersection, then the task will be assigned that location, otherwise a new node is added to the graph (e.g., *Node 14* for *task j3-2*). Also, nodes *4* and *17* represent both an intersection between road segments and the depots. The numbers on the edges represent the segments' length (regular font), and their capacity (subscript).

The problem, using the above defined notation, is then:

$$\mathcal{N} = \{1, \ldots, 20\},\ \mathcal{N}H = \{4, 17\}, \mathcal{O} = \{4, 17\}$$

$$\mathcal{E} = \{(1,2), (2,3), (3,4), (5,6), (6,7), (7,8), (9,10),$$

$$(1,12), (12,13), (17,18), (18,19), (19,20), (1,9),$$

$$(2,5), (3,8), (6,10), (7,13), (8,16), (9,17), (11,15),$$

$$(15,18), (16,19), (14,21)\}$$

$$\mathcal{J} = \{j1, j2, j3, j4\}$$

$$\mathcal{K} = \{i1, i2 \mid \forall i \in \mathcal{J}\}$$

$$L_{j11} = 15,\ L_{j12} = 10,\ L_{j21} = 11,\ L_{j22} = 8,$$

$$L_{j31} = 6,\ L_{j32} = 14,\ L_{j41} = 2,\ L_{j42} = 16$$

$$\mathcal{P}_{j11} = \varnothing, \mathcal{P}_{j12} = j11, \mathcal{P}_{j21} = \varnothing, \mathcal{P}_{j22} = j21$$

$$\mathcal{P}_{j31} = \varnothing, \mathcal{P}_{j32} = j31, \mathcal{P}_{j41} = \varnothing, \mathcal{P}_{j42} = j41$$

$$l_{j11} = 0,\ l_{j12} = 70,\ l_{j21} = 0,\ l_{j22} = 300,$$

$$l_{j31} = 0,\ l_{j32} = 180,\ l_{j41} = 0,\ l_{j42} = 200$$

$$u_{j11} = T,\ u_{j12} = 120,\ u_{j21} = T,\ u_{j22} = 330,$$

$$u_{j31} = T,\ u_{j32} = 250,\ u_{j41} = T,\ u_{j42} = 300$$

$$S_{j11} = 10,\ S_{j12} = 30,\ S_{j21} = 20,\ S_{j22} = 30,$$

$$S_{j31} = 10,\ S_{j32} = 30,\ S_{j41} = 30,\ S_{j42} = 20,$$

$$\mathcal{V} = \{v1, v2, v3, v4\}$$

$\mathcal{V}_{j1} = \{v1, v2\},\ \mathcal{V}_{j2} = \{v1\},$

$\mathcal{V}_{j3} = \{v2, v4\},\ \mathcal{V}_{j4} = \{v3, v4\}$

$OR = 270, C = 3,\ D = 1,\ \rho = 1,\ v = 1$

$T = 500$

First, ComSat will compute the distance for each pair of nodes where either a task or a depot is located. Using the distances, as well as the speed of the vehicles, and all the other parameters, a set of routes CR is computed. In this specific problem, a possible set of routes is:
$R1$: $D1$-$j11$-$j12$-$D1$; Total length: 135; latest start: 42.5; Eligible vehicles: $v1$, $v2$
$R2$: $D1$-$j21$-$j22$-$D1$; Total length: 220; latest start: 200; Eligible vehicles: $v1$
$R3$: $D2$-$j31$-$j32$-$D2$; Total length: 257.5; latest start: 102.5; Eligible vehicles: $v2$, $v4$
$R4$: $D2$-$j41$-$j42$-$D2$; Total length: 195; latest start: 195; Eligible vehicles: $v3$, $v4$

Note that for each route the latest start time is computed based on the strictest time window. Subsequently vehicles are assigned to the routes and an actual start time for the route is given. A possible assignment for the current routes would be $R1$: $v2$; $R2$: $v1$; $R3$: $v4$; $R4$: $v3$. The start time of all routes is 0.

Finally, the routes are *capacity checked*, and this also produces a node-by-node schedule, i.e., the arrival time of a vehicle at each node that is included in the route:
$v1$**:** $v1$-17:0; $v1$-18:18.5; $v1$-15:40.5; $v1$-11:53; $v1$-12:85.5; $v1$-13:98; $v1$-7:153; $v1$-8:300; $v1$-7:347.5; $v1$-6:360; $v1$-10:372.5; $v1$-9:402.5; $v1$-17:440;
$v2$**:** $v2$-17:0; $v2$-18:19.5; $v2$-15:29.5; $v2$-11:52; $v2$-12:64.5; $v2$-10:79.5; $v2$-9:139.5; $v2$-17:177;
$v3$**:** $v3$-4:0; $v3$-3:16; $v3$-2:58.5; $v3$-1:76; $v3$-2:103.5; $v3$-5:116; $v3$-6:128.5; $v3$-7:141; $v3$-8:170.5; $v3$-16:210.5; $v3$-8:270.5; $v3$-3:285.5; $v3$-4:300.5;
$v4$**:** $v4$-4:0; $v4$-3:15; $v4$-8:30; $v4$-7:142; $v4$-8:169.5; $v4$-3:184.5; $v4$-4:199.5; $v4$-14:250; $v4$-4:330.

If the charging coefficient C is increased from 3 to 9, both $R1$ and $R2$ could be assigned to vehicle $v1$, since the charging time would be short enough to allow the vehicle to execute the first route, go back to the depot, recharge and

execute the second route without breaking any time windows. In this case there would still be four routes but three vehicles would be enough to execute them. Moreover, if the operating range of the vehicles were increased to 290, all customers could be served with only three routes:
$R1$: $D1$-$j21$-$j22$-$D1$; Total length: 220; latest start: 200; Eligible vehicles: $v1$
$R2$: $D2$-$j11$-$j12$-$j31$-$j32$-$D2$; Total length: 280; latest start: 100; Eligible vehicles: $v2$
$R3$: $D2$-$j41$-$j42$-$D2$; Total length: 195; latest start: 195; Eligible vehicles: $v3$, $v4$

The solutions presented in this section have been computed using ComSat; a discussion on the algorithm's solving procedure is provided in the next section, after the algorithm itself has been described.

3.2 State Space Analysis

Although the state space size of CF-EVRP is not directly proportional to the solving time, analyzing the state space growth provides an idea of the complexity of the problem itself. The parameters needed to analyze the state space are the number of tasks $|\mathcal{K}|$, the number of nodes $|\mathcal{N}|$ in the graph representing the plant, the number of vehicles $|\mathcal{V}|$, the time horizon T, and the operating range of the vehicles OR. To be able to compute the state space size, the domains of the real valued variables are, in this analysis, restricted to integers.

Based on the model formulation from [26], if capacity constraints are relaxed, the size of the state space is upper bounded by

$$OR^{|\mathcal{V}|\cdot|\mathcal{K}|} \cdot T^{|\mathcal{V}|\cdot|\mathcal{K}|} \cdot 2^{|\mathcal{V}|\cdot|\mathcal{K}|^2}, \tag{D.1}$$

which can be rewritten as

$$2^{|\mathcal{V}|\cdot|\mathcal{K}|\cdot(|\mathcal{K}|+\log_2(OR\cdot T))}. \tag{D.2}$$

Based on the monolithic model of CF-EVRP from [16], if capacity constraints are considered, the size of the state space is upper bounded by

$$2^{|\mathcal{V}|\cdot|\mathcal{K}|} \cdot 2^{|\mathcal{V}|\cdot|\mathcal{K}|\cdot T} \cdot OR^{|\mathcal{V}|\cdot T} \cdot 2^{2\cdot|\mathcal{V}|\cdot|\mathcal{N}|\cdot T}, \tag{D.3}$$

which can be rewritten as

$$2^{|\mathcal{V}|\cdot(|\mathcal{K}|+T(2\cdot|\mathcal{N}|+\log_2(OR)+|\mathcal{K}|))}. \tag{D.4}$$

Assuming that, for a fixed number of vehicles there exists a correlation between the number of tasks and the time horizon, we can define $\beta \in [0,1]$ such that $|\mathcal{K}| = \beta \cdot T$. The ratio between (D.4) and (D.2) then has a dominant factor of $2^{2\cdot|\mathcal{V}|\cdot|\mathcal{N}|\cdot T}$ that arises due to the capacity constraints. Even for small problem instances, this number can be very large. A compositional approach that iteratively solves smaller sub-problems potentially avoids such state space explosion.

4 Problem Decomposition and Solving Procedure

This section describes how the CF-EVRP is decomposed into sub-problems. The first step is a computation of the shortest paths to connect each pair of tasks. In the *Routing Problem* the paths are used to compute routes that start and end at the depots and serve all tasks within their time windows. Once the routes are computed, the *Assignment Problem* matches them with the vehicles, which determines their execution times. Finally, the *Capacity Verification Problem* checks the routes and their execution times against the capacity constraints. It may be required to explore different paths to connect two tasks of a route; this is done by solving the *Paths Changing Problem* that, based on the paths used so far, will find new unexplored paths to connect the tasks. When new paths are computed, the algorithm verifies whether the routes still meet the time windows by solving the *Routes Verification Problem*. Table 1 summarizes inputs and outputs of the algorithms developed to solve each of the sub-problems presented in the following sections.

Table 1: Algorithms to solve the sub-problems of Section 4. Given within parentheses next to the name of the algorithm is the problem that it solves.

Router (*R*outing Problem)
Input: *CP*, *PR* **Output:** *CR* Define optimization problem using (D.5)-(D.19) Optimize and extract *CR* from the solution
Assign (*A*ssignment Problem)
Input: *CR*, *PA* **Output:** *CA* Define feasibility problem using (D.20)-(D.25) Solve and extract *CA* from the solution
CapacityVerifier (*C*apacity Verification Problem)
Input: *CA* **Output:** *CVS* Define feasibility problem using (D.26)-(D.32) Solve and extract *CVS* from the solution
PathsChanger (*P*aths Changing Problem)
Input: *PP* **Output:** *NP* Define optimization problem using (D.33)-(D.39) Optimize and extract *NP* from the solution
RoutesVerifier (*R*outes Verification Problem)
Input: *CR*, *NP* **Output:** *True* or *False* Define feasibility problem using (D.40)-(D.43) Solve problem and return *True* if feasible, else *False*

4.1 Computation of Shortest Paths

In the first step, the shortest path between any two tasks is computed. This is done because the *Routing Problem* is a VRPTW plus additional constraints on tasks precedence and routes length; in a VRP there exist exactly one path between any two tasks. On the other hand, in the CF-EVRP there may be several ways to travel from one task to another. Moreover, computing the shortest paths (instead of any *non-cyclic* path) is required for the algorithm to be sound and complete (see Section 5). The shortest paths *SP* are computed using Dijkstra's algorithm [37] and assigned to the set *CP*. Since each task's location is reachable from any other task's location, a path always exists; also, in case more solutions with the same cost exist, these will be explored in the *Paths Changing Problem*, if needed.

4.2 Routing Problem

Solving the *Routing Problem* means to find a set of routes, i.e., a sequence of tasks' locations that begins and ends at the same depot, such that every task is served within its time windows and the length of each route does not exceed the operating range. This way, once vehicles are assigned to routes in the *Assignment Problem*, they can execute them without having to recharge. Additionally, the routes have to meet the constraints on the tasks' precedence, as introduced above and described below. At this stage capacity constraints are not considered, nor is the actual plant layout. Also, upper bounds on the number of available vehicles are neglected, i.e., there can be more routes than vehicles, since one vehicle can be assigned to more than one route.

With some abuse of notation we can define the distance between two arbitrary tasks' locations as $d_{k_1k_2}$ instead of $d_{L_{k_1}L_{k_2}}$, $\forall k_1 \in \mathcal{K}_{j_1},\ k_2 \in \mathcal{K}_{j_2},\ j_1, j_2 \in \mathcal{J}$. Also, let M_j be the set of mutually exclusive jobs for job j (i.e. vehicles eligible for job j are not eligible for any of the jobs in M_j due to requirements on the vehicle type); let $Perm_j$ be the set of permutations of tasks belonging to job j, where each element *ord* in $Perm_j$ is an ordered list of tasks and let k_{next} be the task coming after task k in *ord*.

The set of decision variables used to build the model for the *Routing Problem* are:

$\theta_{k_1 k_2}$: Boolean variable that is true if a vehicle travels from the location of task k_1 to the location of task k_2, false otherwise.

γ_k: non-negative real variable that models the arrival time of a vehicle at the location of task k.

ϵ_k: non-negative real variable that models the remaining charge of a vehicle when it arrives at the location of task k.

If the solution of the *Routing Problem* turns out to be inconsistent with the vehicles' assignments or the capacity constraints, a new solution must be computed in order to find alternative routes for the same combination of paths. Therefore it is necessary to keep track of the combinations of routes that have already been generated so these can be ruled out when solving the *Routing Problem* again. Let the optimal solution to the *Routing Problem* found at iteration h be $CR = \bigcup_{k_1,k_2 \in \mathcal{K}} \{\theta^*_{k_1 k_2}\}$, where $\theta^*_{k_1,k_2}$, $\forall k_1, k_2 \in \mathcal{K}$, is the value of θ_{k_1,k_2} in the current solution; also, let PR be the set containing the optimal solutions found until the $(h-1)$-th iteration.

The following logical operators are used to express cardinality constraints [38] in the sub-problems:

$\mathrm{EN}(a, n)$: exactly n variables in the set a are true;

$\mathrm{If}(c, o_1, o_2)$: if c is *true* returns o_1, else returns o_2.

To shorten the notation we will write $\mathrm{EN}_{m \in M}(m, n)$ to denote $\mathrm{EN}(\bigcup_{m \in M} \{m\}, n)$. The model formulation for the *Routing Problem* is as follows:

$$\min_{k\in\mathcal{K}_j,\ j\in\mathcal{J},\ s\in\mathcal{S}} \sum \mathrm{If}(\theta_{sk}, 1, 0) \tag{D.5}$$

$$\epsilon_k \cdot \rho \leq OR, \qquad \forall k \in \mathcal{K}_j,\ j \in \mathcal{J} \tag{D.6}$$

$$\neg\theta_{kk}, \qquad \forall k \in \mathcal{K}_j,\ j \in \mathcal{J} \tag{D.7}$$

$$\neg\theta_{ks}, \qquad \forall k \in \mathcal{K}_j,\ s \in \mathcal{S},\ j \in \mathcal{J} \tag{D.8}$$

$$\neg\theta_{fk}, \qquad \forall f \in \mathcal{F},\ k \in \mathcal{K}_j,\ j \in \mathcal{J} \tag{D.9}$$

$$\theta_{k_1k_2} \implies \gamma_{k_2} \geq \gamma_{k_1} + S_{k_1} + d_{k_1k_2}/v, \qquad \forall k_1 \in \mathcal{K}_{j_1},\ k_2 \in \mathcal{K}_{j_2},\ j_1, j_2 \in \mathcal{J} \tag{D.10}$$

$$\theta_{k_1k_2} \implies \epsilon_{k_2} \leq \epsilon_{k_1} - D \cdot d_{k_1k_2}/v, \qquad \forall k_1 \in \mathcal{K}_{j_1},\ k_2 \in \mathcal{K}_{j_2},\ j_1, j_2 \in \mathcal{J} \tag{D.11}$$

$$\mathrm{EN}_{\substack{k_2\in\mathcal{K}_{j_2},\\ j_2\in\mathcal{J}}}(\theta_{k_1k_2}, 1), \qquad \forall k_1 \in \mathcal{K}_{j_1},\ j_1 \in \mathcal{J},\ j_1 \neq j_2 \tag{D.12}$$

$$\mathrm{EN}_{\substack{k_1\in\mathcal{K}_{j_1},\\ j_1\in\mathcal{J}}}(\theta_{kk_1}, n) \implies \mathrm{EN}_{\substack{k_2\in\mathcal{K}_{j_2},\\ j_2\in\mathcal{J}}}(\theta_{k_2k}, n), \qquad \forall j \in \mathcal{J},\ k \in \mathcal{K}_j,\ n = 1, \ldots, |\mathcal{J}| \tag{D.13}$$

$$\mathrm{EN}_{\substack{k_1\in\mathcal{K}_{j_2},\\ j_2\in\mathcal{J}}}(\theta_{k_1k}, n) = \mathrm{EN}_{\substack{k_2\in\mathcal{K}_{j_2},\\ j_2\in\mathcal{J}}}(\theta_{kk_2}, n), \qquad \forall k \in \mathcal{S} \cup \mathcal{F},\ n = 1, \ldots, |\mathcal{J}| \tag{D.14}$$

$$\gamma_k \geq l_k \wedge \gamma_k \leq u_k, \qquad \forall k \in \mathcal{K}_j,\ j \in \mathcal{J} \tag{D.15}$$

$$\neg\theta_{k_1k_2}, \quad \forall k_1 \in \mathcal{K}_{j_1},\ k_2 \in \mathcal{K}_{j_2},\ j_1 \in \mathcal{J},\ j_2 \in M_{j_1} \tag{D.16}$$

$$\bigvee_{ord\in Perm_j} \left(\bigwedge_{k\in ord} \theta_{kk_{\mathrm{next}}} \right), \qquad \forall j \in \mathcal{J} \tag{D.17}$$

$$\bigwedge_{k'\in\mathcal{P}_k} \gamma_k \geq \gamma_{k'}, \qquad \forall k \in \mathcal{K} \tag{D.18}$$

$$\bigvee_{\theta_{k_1k_2}\in\lambda} \neg\theta_{k_1k_2}, \qquad \forall \lambda \in PR \tag{D.19}$$

The cost function to minimize (D.5) is the total number of routes. This is done by minimizing the number of direct travels from the tasks representing the depots; (D.6) restricts the remaining charge to be lower than or equal to the maximum operating range; (D.7) forbids to travel from and to the same location; (D.8) and (D.9) express that a vehicle can never travel to the start, nor

travel from the end: *start* and *end* referring to the same depot are physically located at the same node, but they play different roles in the *Routing Problem*, hence two different tasks; (D.10) constrains the difference in arrival time based on the distance for a direct travel between two points; (D.11) models the decrease of charge based on the distance travelled. For (D.10) and (D.11) distances are computed using the current paths CP; (D.12) expresses that each task's location must be visited exactly once; (D.13) guarantees the flow conservation between start and end; (D.14) ensures that all vehicles leaving the depots return after visiting the tasks' locations; (D.15) enforces the time on the routes; (D.16) expresses that there cannot be direct travel among mutual exclusive jobs. This constraint does not always guarantee that mutually exclusive jobs will never be executed in the same route. Some corner cases are not covered, but the inconsistency will be spotted in the *Assignment Problem* so there is no need to further complicate the constraint (there would be need to enumerate a large number of task sequences and rule out the inconsistent ones by adding one constraint for each of them), since it would slow down the whole sub-problem solution; (D.17) expresses that if a number of tasks belong to one job, they have to take place in sequence; (D.18) guarantees that precedence constraints among tasks are enforced. Constraint (D.19) allows to rule out the previously computed sets of routes as a solution. This is necessary as this optimization sub-problem may be called multiple times during the execution of ComSat.

Based on the model described above, the algorithm *Router* is defined, that takes the set of current paths CP and the set PR, and returns a set of routes CR that have not been selected yet; if the problem is infeasible, CR is empty.

4.3 Assignment Problem

The routes CR are generated in the *Routing Problem* based only on the time windows and on the vehicles' operating range. The *Assignment Problem* now allocates vehicles to the routes based on the actual availability of each type of vehicle. Moreover, even though constraint (D.16) partially prevents it, CR may contain routes that involve mutually exclusive jobs and, while it would be possible to avoid this by adding additional constraints, it would be inconvenient to do so in the *Routing Problem*, since there is no information about the vehicles assigned to the routes. On the other hand, once a set of routes is given, it is verified in the *Assignment Problem* whether a vehicle is actually eligible for a route.

Therefore for each route $r \in CR$, we can define a list of jobs $\mathcal{J}^r \subseteq \mathcal{J}$ that are executed by the vehicle assigned to r, and the list of eligible vehicles for r, $El_r = \bigcap_{j \in \mathcal{J}^r} \mathcal{V}_j$. Also, based on the time windows and service times of the jobs forming the routes, it is possible to work out the latest start of a route $late_r$. Since a route can include more than one job, the strictest time window will define the latest start for the route. Finally, for each route we can define the cumulative service time $S_r = \bigcup_{\substack{k \in j \\ j \in \mathcal{J}^r}} S_k$.

The *Assignment Problem* is formulated as a JSP where routes are jobs (whose durations depend on their lengths $length_r$) and vehicles are resources, with some additional requirements on the jobs' starting time. The set of decision variables used to build the model are:

α_{ir}: Boolean variable that is true if vehicle i is assigned to route r, false otherwise;

s_r: non-negative real variable that models the start time of route r;

e_r: non-negative real variable that models the end time of route r.

It may be necessary to have different assignments for the same set of routes CR, since two vehicles located at the same depot may have different states of charge and, therefore, lead to different outcomes when solving the *Capacity Verification Problem*. Thus, let the optimal solution to the *Assignment Problem* found at iteration h be $CA = \bigcup_{\substack{i \in \mathcal{V} \\ r \in CR}} \{\alpha^*_{ir}\}$, where α^*_{ir}, $\forall i \in \mathcal{V}$, $r \in CR$, is the value of α_{ir} in the current solution; also, let PA contain the optimal solutions found until the $(h-1)$-th iteration.

The model formulation for the *Assignment Problem* is:

$$\mathrm{EN}_{i \in V}(\alpha_{ir}, 1), \qquad \forall r \in CR \tag{D.20}$$

$$e_r = s_r + \mathit{length}_r / v + S_r, \qquad \forall r \in CR \tag{D.21}$$

$$s_r \leq \mathit{late}_r, \qquad \forall r \in CR \tag{D.22}$$

$$\bigvee_{i \in El_r} \alpha_{ir}, \qquad \forall r \in CR \tag{D.23}$$

$$(\alpha_{ir} \wedge \alpha_{ir'}), \implies$$
$$(s_r \geq e_{r'} + C \cdot \mathit{length}_r) \vee (s_{r'} \geq e_r + C \cdot \mathit{length}_{r'}),$$
$$\forall i \in V,\ r, r' \in CR,\ r \neq r' \tag{D.24}$$

$$\bigvee_{\alpha_{ir} \in \lambda} \neg \alpha_{ir} \qquad \forall \lambda \in PA \tag{D.25}$$

Constraint (D.20) guarantees that exactly one vehicle is assigned to each route; (D.21) connects the *start* and *end* variables based on the route's length and their cumulative service time; (D.22) constrains the latest start time of a route to the strictest time window of its jobs; (D.23) expresses that one (or more) among the eligible vehicles must be assigned to a route; (D.24) expresses that any two routes assigned to the same vehicle cannot overlap in time; one must end before the other starts. Finally, constraint (D.25) guarantees to find an assignment different from the already found ones.

Based on the model described above, the algorithm *Assign* is defined, that takes the set of current routes CR from the routing problem as input, and returns the current assignment CA that specifies which vehicle that will use each route (and execute its jobs) and when it starts; if the *Assignment Problem* is infeasible, $CA = \varnothing$.

4.4 Capacity Verification Problem

In this phase the goal is to find a feasible schedule for the vehicles, if it exists, meaning that the routes they are assigned to are evaluated to verify that capacity constraints are fulfilled. To do this, an ordered list of nodes NL_r and an ordered list of edges EL_r, $\forall r \in CR$, respectively, are generated, that each route visits. Let n_{re} be the node visited before edge e on route r and let e_{rn} be the node visited before node n on route r. Similarly, let n^{re} be the node visited after edge e on route r and let e^{rn} be the node visited after node n on route r. Also, for each node in NL_r it is necessary to specify whether there exists a time window, since some of the nodes are only intersections of road segments in the real plant, while others are actual pickup or delivery points. Let l_{rn} and u_{rn} be the earliest and latest arrival time, respectively, at node n on route r; let S_{rn} be the service time at node n on route r, if such exists, *zero* otherwise. Let n_r^* be the starting node of route r. Finally, let $e(1)$ and $e(2)$ be the source and sink node of edge e respectively.

This phase is also treated as a JSP, where routes are jobs, while nodes and edges are the resources. Also, each route has a starting time s_r defined by solving the *Assignment Problem*.

The decision variables in the *Capacity Verification Problem* are:

x_{rn}: non-negative real variables to model when route r is using node n;

y_{re}: non-negative real variable to model when route r is using edge e;

The model for the *Capacity Verification Problem* is:

$$x_{rn_r^*} \geq start_r, \qquad \forall r \in CR \tag{D.26}$$

$$y_{re} \geq x_{rn_{re}} + S_{rn_{re}}, \qquad \forall r \in CR,\ e \in EL_r \tag{D.27}$$

$$x_{rn} = y_{re_{rn}} + d_{e_{rn}}, \qquad \forall r \in CR,\ n \in NL_r \tag{D.28}$$

$$x_{rn} \geq l_{rn} \wedge x_{rn} \leq u_{rn}, \qquad \forall r \in CR,\ n \in NL_r \tag{D.29}$$

$$\begin{aligned} x_{r_1 n} \geq y_{r_2 e^{r_1 n}} + 1 \ \vee \ x_{r_2 n} \geq y_{r_1 e^{r_2 n}} + 1, \\ \forall r_1, r_2 \in CR,\ r_1 \neq r_2, \\ n \in NL_{r1} \cap NL_{r2},\ n \notin \mathcal{N}H \end{aligned} \tag{D.30}$$

$$\begin{aligned} y_{r_1 e} \geq y_{r_2 e} + 1 \vee y_{r_2 e} \geq y_{r_1 e} + 1, \\ \forall r_1, r_2 \in CR,\ r_1 \neq r_2,\ e \in EL_{r_1} \cap EL_{r_2},\ g(e) = 1 \end{aligned} \tag{D.31}$$

$$\begin{aligned} y_{r_1 e_1} \geq y_{r_2 e_2} + d_{e_2} \ \vee y_{r_2 e_2} \geq y_{r_1 e_1} + d_{e_1}, \\ \forall r_1, r_2 \in R,\ r_1 \neq r_2,\ e_1 \in EL_{r_1}, \\ e_2 \in EL_{r_2},\ e_1 = \bar{e}_2,\ g_{e_1} = g_{e_2} = 1 \end{aligned} \tag{D.32}$$

(D.26) constraints the start time of a route; (D.27) and (D.28) define the precedence among nodes and edges to visit in a route; (D.29) enforces time windows on the nodes that correspond to the tasks; (D.30) prevents vehicles from using the same node at the same time (the +1 in the constraints forbids *swapping* of positions between a node and the previous or following edge); (D.31) and (D.32) constrain the transit of vehicles over the same edge. If two vehicles are using the same edge from the same node, one has to start at least one time-step later than the other and if two vehicles are using the same edge from opposite nodes, one has to be done transiting, before the other one can start.

Based on the model described above, the algorithm *CapacityVerifier* is defined, that takes the *CA* from the *Assignment Problem* as input and returns *CVS*, a list that expresses where each vehicle is at each time-step and, as for the previous phases, is empty if the problem is infeasible.

4.5 Paths Changing Problem

In this phase, alternative paths are computed to connect the consecutive tasks of each route. Finding alternative paths may be necessary when, for a given set of routes CR, it is not possible to find any feasible schedule CVS. The infeasibility of the *Capacity Verification Problem* may be due to the current set of paths CP that connect the tasks' locations, therefore a different set may lead to a feasible solution. We have previously defined a route as a sequence of tasks' locations and for any two consecutive tasks there is a path (a sequence of edges) connecting them. Therefore, for a route counting $i + 1$ tasks we will have i paths and for each path we can define a start and an end node, respectively ξ_i and π_i. Finally, we define the sets of outgoing and incoming edges for a certain node n as $\mathcal{U}_n$ and $\mathcal{I}_n$, respectively.

Variables used to build the model are:

w_{rin}: Boolean variable that represents whether the i-th path of route r is using node n;

z_{rie}: Boolean variable that represents whether the i-th path of route r is using edge e;

We could split this problem into $r \cdot i$ problems (assuming all routes have $i+1$ tasks) and find paths for each route separately; simpler models are faster. Unfortunately it may be necessary to explore different combinations of paths and so to retain the information we need we have only one model. Therefore, let the optimal solution to the *Path Changing Problem* found at iteration h be

$$CP = \bigcup_{\substack{r \in CR \\ i=1,\dots,|r| \\ e \in \mathcal{E}}} \{z^*_{rie}\},$$

where z^*_{rie}, $\forall i = 1, \dots, |r|,\ r \in CR,\ e \in \mathcal{E}$, is the value of z_{rie} in the current solution; also, let PP be the set containing the optimal solutions found until the $(h-1)$-th iteration.

The model, similar to [39], is as follows:

$$\min_{i=1,..,|r|,\ r\in CR,\ n\in\mathcal{N}} \sum \mathrm{If}(w_{rin}, 1, 0) \tag{D.33}$$

$$w_{ri\xi_i} \wedge w_{ri\pi_i}, \qquad \forall i = 1,\dots,|r|,\ r \in CR \tag{D.34}$$

$$\mathrm{EN}_{e\in\mathcal{U}_{\xi_i}}(z_{rie}, 1), \qquad \forall i = 1,\dots,|r|,\ r \in CR \tag{D.35}$$

$$\mathrm{EN}_{e\in\mathcal{I}_{\xi_i}}(z_{rie}, 1), \qquad \forall i = 1,\dots,|r|,\ r \in CR \tag{D.36}$$

$$z_{rie} \implies \neg z_{ri\bar{e}}, \qquad \forall i = 1,\dots,|r|,\ r \in CR,\ e \in \mathcal{E} \tag{D.37}$$

$$\bigwedge_{n\in\mathcal{N}, n\neq\xi_i, n\neq\pi_i} \mathrm{If}(w_{rin}, \mathrm{EN}_{e\in\mathcal{U}_n}(z_{rie}, 1) \wedge \mathrm{EN}_{e\in\mathcal{I}_n}(z_{rie}, 1), \mathrm{EN}_{e\in\mathcal{U}_n}(z_{rie}, 0) \wedge \mathrm{EN}_{e\in\mathcal{I}_n}(z_{rie}, 0)), \quad \forall i = 1,\dots,|r|,\ r \in CR \tag{D.38}$$

$$\bigvee_{z_{rie}\in CP} \neg z_{rie}, \qquad \forall CP \in PP \tag{D.39}$$

The cost function (D.33) to minimize is the total number of used edges; (D.34) guarantees that, for each path of each route, the start and end nodes are used; (D.35) and (D.36) make sure that exactly one outgoing (incoming) edge is incident with the start (end) node of a route; (D.37) makes sure that a path cannot use both an edge and its reverse; (D.38) guarantees that if a node (different from the start or end) is selected, exactly one of its outgoing and one of its incoming edges will be used. On the other hand, if a node is not used, none of its incident edges will be used; finally, (D.39) rules out all the previously found solutions.

Based on the model described above the algorithm *PathsChanger* is defined, that takes the previous paths PP as input and returns a new set of paths NP, such that $NP \cap PP = \varnothing$. If the *Paths Changing Problem* is infeasible $NP = \varnothing$.

4.6 Routes Verification Problem

The *Routes Verification Problem* is a simplified version of the *Routing Problem*, where a set of routes CR already exists and it is verified whether these meet the requirements on the tasks' time windows and the vehicles' operating range. As described in Section 4.2, routes are designed based (among other things) on the distance between tasks' locations; paths are computed between any two tasks' locations to have a uniquely defined distance and the routes designed accordingly in the *Routing Problem*. However, as soon as the paths used to connect the tasks' locations are changed, there is no guarantee that the routes still meet the requirements, hence the need to verify the routes.

Let $\mathcal{K}_r = \{k_1, \ldots, k_{|r|}\}$ be the set of task for route $r \in CR$, the variables used to build the model for the *routes verification problem* are:

> σ_{rk}: non-negative real variable that models the time when task k of route r is served
>
> ω_{rk}: non-negative real variable that models the remaining charge of a vehicle assigned to route r when it reaches task k

The model is as follows:

$$\omega_{rk} \cdot \rho \leq OR, \qquad \forall k \in \mathcal{K}_r,\, r \in CR \tag{D.40}$$

$$\sigma_{rk_{i+1}} \geq \sigma_{rk_i} + d_{k_i k_{i+1}} + S_{k_i}, \qquad \forall i = 1, \ldots, |r|,\, r \in CR \tag{D.41}$$

$$\sigma_{rk} \geq l_k \wedge \sigma_{rk} \leq u_k, \qquad \forall k \in \mathcal{K}_j,\, j \in \mathcal{J} \tag{D.42}$$

$$\omega_{rk_{i+1}} \leq \omega_{k_i} - D \cdot d_{k_i k_{i+1}}, \qquad \forall i = 1, \ldots, |r|,\, r \in CR \tag{D.43}$$

(D.40) restricts the domain of the remaining charge to be smaller than or equal to the operating range of the vehicles; (D.41) connects the arrival time at the task based on the distance between them; (D.42) forces the arrival time at a task's location to be within its time window; (D.43) relates the remaining charge when reaching a task's location to the distance from the previous task's location.

Based on the model described above the algorithm *RoutesVerifier* is defined, that takes the current routes CR and the current paths CP and returns true if the problem is feasible, false otherwise.

4.7 Solving the CF-EVRP using the ComSat algorithm

The *ComSat* algorithm, Figure 2, connects the above described sub-problems to find a feasible solution to the full problem. The *Router* and *PathsChanger* algorithms are in Figure 2 put in rounded corner boxes, to show that they are optimization problems.

The algorithm begins with the computation of the shortest paths between each pair of tasks. This step is only executed once to provide unique paths for the *Routing Problem*, which is then solved. In this step neither the vehicles' availability nor the segment capacities are considered; the goal is simply to design routes to serve tasks within the time windows. Therefore, if the *Routing Problem* is infeasible, the whole problem is infeasible, because there is no possible routing such that tasks are served within their time windows. The information about the previous routes will be stored so that each time this algorithm is called, it will provide a new solution to the *Routing Problem*.

If the *Routing Problem* is feasible the next step is to verify whether the available vehicles can execute the routes. This matching is based on the routes' requirements for specific types of vehicles, on their latest start time, and on the vehicles' operating range and charge rate. This is done by solving the *Assignment Problem*; also in this case there can be more feasible solutions, therefore it is important to store the current one to be able to rule it out the next time the *Assignment Problem* is solved. If the *Assignment Problem* is infeasible the algorithm backtracks and runs the *Routing Problem* again, otherwise, it proceeds to the *Capacity Verification Problem*.

At this point, routes have been assigned an actual vehicle to execute them and start times have been restricted to meet the vehicles' need for charging. Hence it is possible to verify if the execution of the routes is possible without breaking the capacity constraints. If that is the case, the overall problem is feasible and the algorithm terminates and returns a feasible schedule. On the other hand, if this step is infeasible, the algorithm will try to find alternative paths for the vehicles to execute the routes.

This step is split in two parts. The *PathsChanger* algorithm finds new paths and the *RoutesVerifier* makes sure the *Routing Problem* is still solvable (i.e. tasks can still be served within time windows) using these new paths. If the *Paths Changing Problem* is infeasible, all paths from one task to the following one have been checked for each route. Therefore the algorithm backtracks and looks for a assignment. Otherwise if the *Paths Changing Problem* is

feasible, the algorithm moves forward to the *Routes Verification Problem*. If this problem is feasible the algorithm backtracks to verify whether it is feasible against the capacity constraint by solving the *Capacity Verification Problem*; if not, the *PathsChanger* algorithm is called again.

Whenever the *Assignment Problem* is infeasible, all possible assignments for the current set of routes *CR* have been explored. Thus, before calling the *Router* algorithm again, *CR* is added to *PR*. In the same way, whenever the *Paths Changing Problem* is infeasible, all possible paths for the current assignment *CA* have been explored, hence *CA* is added to *PA*. Also, the set of previous paths *PP* is emptied because these paths are only eligible for the current assignment, and the shortest paths are set as current paths to compute the next assignment.

Table 2: Glossary for the sets of the sub-problems.

CP: set of current paths
SP: set of shortest paths
NP: set of new paths
PP: set of previous paths
CR: set of current routes
PR: set of previous routes
CA: set of current assignment of vehicles to routes
PA: set of previous assignment of vehicles to routes
CVS: set of *conflict-free* routes (pairs of nodes and arrival times for each route)
RVF: Boolean variable representing the feasibility of the *Routing Problem*

On the other hand, when exploring different paths the current assignment is not changed, it is only checked whether it is feasible with the new paths; thus, *CA* is not added to *PA*. Finally, as ComSat loops through *PathsChanger* and *RoutesVerifier* to find a feasible set of paths, *NP* is assigned to *CP*, which in turn is added to *PP* after every unsuccessful iteration. The glossary of Table 2 contains the names of the sets used to store and exchange information among the sub-problems presented in Section 4.

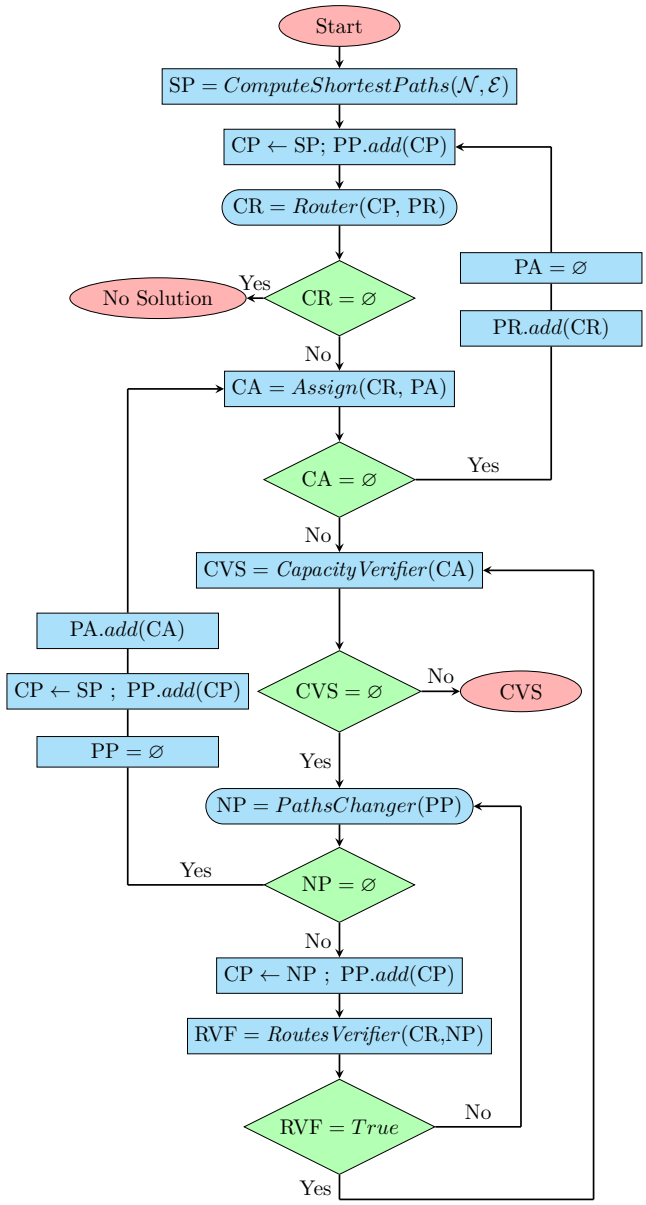

Figure 2: Flowchart of ComSat algorithm.

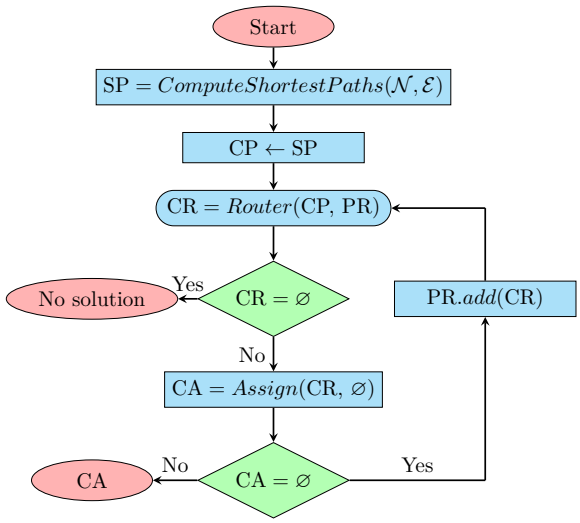

Figure 3: Flowchart of C-ComSat.

4.8 Solving the Example using ComSat

To illustrate the performance of ComSat the example of Section 3 is used. Below are reported the function calls to solve the sub-problems, whether they are feasible or infeasible, and their running times:

Router: *feasible*, 0.45 s

Assign: *feasible*, 0.05 s

CapacityVerifier: *feasible*, 0.08 s

The implementation of ComSat has the ability of manually setting the solution to a sub-problem to be *infeasible*. We set the solution to the *Capacity Verification Problem* to be *infeasible* to evaluate the performance of the algorithm when conflicts arise due to capacity constraints. We also set a limit of 50 on the number of alternative sets of paths to generate. Of these 50 sets, 29 were declared infeasible by the *RoutesVerifier*. The average running time to solve the *Path Changing Problem* is 0.7 s; as for the *Routes Verification Problem*, it took on average 0.01 s to solve it, regardless of the feasibility of the sub-problem.

For the larger problem instances discussed in Section 6 counting up to 11 vehicles and 15 jobs, the *Router* and the *PathChanger* calls take roughly 10 s each, while the *CapacityVerifier* takes about 1.5 s, *Assign* takes less than 0.5 s, and *RoutesVerifier* takes between 0.01 s and 0.1 s.

5 Soundness and Completeness of ComSat

This section provides proof of ComSat's soundness and completeness. We start by relaxing the capacity constraints on the segments. We refer to the relaxed version of ComSat as the *Capacity Relaxed-ComSat* (*C-ComSat*). This turns the problem into a combination of routing and assignment. We use the conclusions from C-ComSat as a starting point to prove the soundness and completeness of ComSat.

When we do not have to deal with capacity constraints we can simplify ComSat, as shown in Fig. 3; we essentially have to solve the *Routing Problem* and then verify that the solution found has a feasible assignment by solving the *Assignment Problem*. If that is the case, the algorithm terminates and returns a feasible solution, else it tries to design different routes. If no routing solution has a feasible assignment, the algorithm terminates with *No Solution*.

Observation 1. *All the problems solved in ComSat are decidable. This is true because they are all combinations of decidable first-order theories and therefore the* Nelson-Oppen theory combination method *[40] applies. In fact the* Routing Problem *is a combination of linear arithmetic and propositional logic, the* Assignment Problem*,* Capacity Verification Problem*, and* Routes Verification Problem *all fall into the category of difference logic (a fragment of linear arithmetic), and the* Paths Changing Problem *is a propositional logic problem.*

Observation 2. *The optimization problems solved in ComSat, i.e., the* Routing Problem *and the* Paths Changing Problem*, are bounded. The* Routing Problem *involves a finite number of decision variables that are either Booleans with a finite domain, or non-negative integers and the objective is minimization. The* Paths Changing Problem *involves only Boolean variables, so the domain is finite.*

Lemma 1: *Given a problem instance of the CF-EVRP, if a feasible solution to the* Routing Problem *cannot be found using the shortest paths to*

connect any two tasks, no feasible solution can be found using any other set of paths.

Proof. The *Routing Problem* can be infeasible for two reasons (or a combination of them): there exist no routes such that all tasks' time windows can be met; there exist no routes shorter than or equal to the vehicles operating range such that all tasks are served.

When it comes to the time windows, the lower bound does not affect the feasibility of an instance, because vehicles are allowed to wait at the task's location before starting the service. On the other hand, if there is no way a vehicle can reach a task's location before the time window's upper bound the instance is infeasible. Since it is assumed that vehicles travel at constant speed, the distances among tasks' locations are directly proportional to the time required to travel between them; therefore, if time windows cannot be met travelling along the shortest paths, neither can they using any other set of paths.

As for the routes maximum length, restricted by the vehicles operating range, the same reasoning applies. If it is not possible to design routes to serve all tasks that are shorter than or equal to the vehicles operating range using the shortest paths, neither will it be using longer paths. □

Lemma 1 is required both for C-ComSat and ComSat. When relaxing the capacity constraints, before ruling out a set of routes it needs to be made sure that they are infeasible. If arbitrarily long paths were used, there would still be a chance that using other paths could make the *Routing Problem* feasible. As for ComSat, if a set of routes is infeasible using the shortest paths, there is no need to try to replace paths and check for feasibility.

5.1 Relaxed Problem: Capacity constraints not included

When relaxing the capacity constraints the problem boils down to designing the routes, which is taken care of by the *Routing Problem*, and the assignment of the available vehicles, handled by solving the *Assignment Problem*.

Lemma 2: *The* Routing Problem *has a finite number of feasible solutions.*

Proof. Let φ be the conjunction of constraints (D.7)-(D.18), let φ^r be the conjunction of constraints (D.7)-(D.14), and let φ^s be the conjunction of constraints (D.15)-(D.18). φ^r does not take into account tasks, jobs, operating

range, or time windows; it only guarantees that routes are closed and that each task is visited exactly once. Let $k = |\mathcal{K}|$ and let a route r be an ordered subset of $\mathcal{K}$. Then, a solution to the routing problem $R = \{ r_1, \ldots, r_m \,|\, m \leq k\}$ is a partition of the set $\mathcal{K}$. The number of partitions of the set $\mathcal{K}$is $\sum_{q=0}^{k} \binom{k}{q} < \infty$, which corresponds to the number of possible solutions for φ^r, $\mathbb{S}(\varphi^r)$. Since $\varphi = \varphi^r \wedge \varphi^s$, then $\mathbb{S}(\varphi) \leq \mathbb{S}(\varphi^r) < \infty$ □

Lemma 3: *Repeated calls to* Router *will enumerate all feasible solutions before returning infeasible.*

Proof. Let φ_0 be a relaxation of the *Routing Problem*, not including constraint (D.18), and let CR_0 be a solution to φ_0. Then, if another solution CR_1 for φ_0 exists, it can be found by solving $\varphi_0 \wedge \neg CR_0 = \varphi_1$. In general, the n-th solution can be found by solving $\varphi_0 \wedge \neg CR_0 \wedge \ldots \wedge \neg CR_{n-1} = \varphi_n$. Because of Lemma 2, we know that $\mathbb{S}(\varphi) < \infty$ and we enumerate all by solving φ_0, …, $\varphi_{\mathbb{S}(\varphi)-1}$ □

Theorem 1: *C-ComSat in Fig. 3 is sound and complete.*

Proof. For a problem where $|\mathcal{K}|$ tasks are to be executed and $|\mathcal{V}|$ vehicles are available, a solution is an assignment $CA = \{(v, r_1)_1, \ldots, (v, r_i)_i\}$ $v \in \mathcal{V}$, $i \leq |\mathcal{K}|$ that satisfies the *Assignment Problem* and $\bigcup_{i \leq |\mathcal{K}|} r_i$ is a feasible solution to the *Routing Problem*. Because of *Lemma* 2 we know there is only a finite number of solutions to the *Routing Problem*, and because of *Lemma* 3 we know we can enumerate them all. In the algorithm, for each solution *CR* we check whether it satisfies the *Assignment Problem*; hence, if the overall problem has a feasible solution, the algorithm will eventually find it, otherwise it will declare the problem infeasible. □

5.2 Full Problem

When including the capacity constraints, the paths chosen to move from one task's location to another become crucial. For ComSat to be sound and complete, we need to prove that it can explore all possible paths for a given set of routes and a given assignment. Note that we have restricted the problem to forbid cycles in the paths; however, in some instances of the CF-EVRP cycles may be required. We are going to show that, if we use a different model to change paths such that also cycles are allowed, the algorithm is sound and complete for any instance of the CF-EVRP.

Lemma 4: *Given a directed, weighted graph with a finite number of nodes, the number of paths that connect two arbitrary nodes is finite.*

Proof. By definition, a path is a sequence of edges that joins a sequence of nodes and no node appears more than once. If the number of nodes in the graph is finite, there cannot be an infinite number of sequences of nodes to connect to arbitrary nodes. □

Lemma 5: *For a given set of routes CR and a given assignment of vehicles CA, repeated calls to the* PathsChanger *algorithm will enumerate all feasible solutions before returning infeasible.*

Proof. Let φ_0 be the conjunction of constraints (D.34)-(D.38), a relaxation of the *Paths Changing Problem*, and let CP_0 be a solution to φ_0. Then, if another solution CP_1 for φ_0 exists, it can be found by solving $\varphi_0 \wedge \neg CP_0 = \varphi_1$. In general, the n-th solution can be found by solving $\varphi_0 \wedge \neg CP_0 \wedge \ldots \wedge \neg CP_{n-1} = \varphi_n$. Because of *Lemma* 4, we know that the number of solutions to the *Paths Changing Problem* $\mathbb{S}(\varphi) < \infty$ and we enumerate all by solving $\varphi_0, \ldots, \varphi_{\mathbb{S}(\varphi)-1}$. □

When considering the capacity constraints, different assignments may lead to different solutions; for instance one vehicle may not be available until a certain time because it is still recharging after executing a route while another is available earlier. Both assignments are feasible but they will execute the routes at different time, hence the road segments will be occupied by the vehicles at different times, which in turn may lead to different schedules. Therefore it is necessary to explore all possible assignments.

Lemma 6: *The* Assignment Problem *has a finite number of feasible solutions.*

Proof. In an instance of the CF-EVRP we assume to have a finite number of vehicles $v = |\mathcal{V}|$. We know from Lemma 2 that, given a finite number of tasks, a solution to the *Routing Problem* has at most as many routes as tasks. Let φ be the conjunction of constraints (D.20)-(D.24), let φ^r be the conjunction of constraint (D.20), and let φ^s be the conjunction of constraints (D.21)-(D.24). φ^r is a relaxation of the *Assignment Problem* that does not take into account routes length, charging time or vehicle eligibility, it only guarantees that exactly one vehicle is assigned to each route. A solution to

the *Assignment Problem*, represented by φ^r, is therefore a partition of the routes set *CR*. Let $c = |CR|$, then the number of partitions of the set *CR* is $\sum_{q=0}^{c} \binom{c}{q} < \infty$, which corresponds to the number of possible solutions for φ^r, $\mathbb{S}(\varphi^r)$. Since $\varphi = \varphi^r \wedge \varphi^s$, then $\mathbb{S}(\varphi) \leq \mathbb{S}(\varphi^r) < \infty$ □

Lemma 7: *For a given set of routes CR, repeated calls to* Assign *will enumerate all feasible solutions before returning infeasible.*

Proof. Let φ_0 be the conjunction of constraints (D.21)-(D.24), a relaxation of the *Assignment Problem*, and let CA_0 be a solution to φ_0. Then, if another solution CA_1 for φ_0 exists, it can be found by solving $\varphi_0 \wedge \neg CP_0 = \varphi_1$. In general, the n-th solution can be found by solving $\varphi_0 \wedge \neg CA_0 \wedge \ldots \wedge \neg CA_{n-1} = \varphi_n$. Because of *Observation* 6, we know that the number of solutions to the *Assignment Problem* $\mathbb{S}(\varphi) < \infty$ and we enumerate all by solving $\varphi_0, \ldots, \varphi_{\mathbb{S}(\varphi)-1}$. □

Theorem 2: *ComSat is sound and complete.*

Proof. For a problem with $|\mathcal{K}|$ tasks, $|\mathcal{V}|$ vehicles, and a graph $G(\mathcal{N}, \mathcal{E})$, let $\overline{r}$ be the sequence of nodes visited to execute route r and τ_{rn} the arrival time at node n of route r; a solution is a schedule for each vehicle $v \in \mathcal{V}$,

$$CVS = \{(v, ((n_{11}, \tau_{11}), \ldots, (n_{1\overline{r}_1}, \tau_{1\overline{r}_1})))_1, \ldots, \\ (v, ((n_{i1}, \tau_{i1}), \ldots, (n_{i\overline{r}_i}, \tau_{i\overline{r}_i})))_i\}, \forall i \leq |\mathcal{K}|,$$

that satisfies the *Capacity Verification Problem*. From *Lemma* 3 we know that we can enumerate all possible routes and from *Lemma* 7 we know that, for each set of routes we can enumerate all assignments. For each assignment (v, r) the arrival time of vehicle v at a node depends on the paths chosen to travel from one task of route r to the following one. Since we know from *Lemma* 5 that for a current set of routes *CR*, and an assignment *CA* of vehicles to it, we can enumerate all paths from one task of each route to the following one, if there exists a solution to the problem, ComSat will eventually find it; otherwise it will correctly declare the problem infeasible. □

As mentioned before, the *PathsChanger* can only return (non-cyclic) paths. We know that there is a finite number of paths in a graph to go from one node to another, this is not true if cycles are allowed. On the other hand, even if cycles were allowed, if we limited the paths' maximum length, we could enumerate them all. Since we have time windows on the tasks and a

limited operating range for the vehicles, we can compute an upper bound for the length of the pahts.

Therefore, if we were to modify the *PathsChanger* to allow for cycles, the algorithm would still be complete without restricting the problem to non-cyclic paths. This feature is currently under investigation as future work.

6 Evaluation

In order to evaluate ComSat, a set of benchmark problems is proposed. The parameters for generating the benchmark problems are the number of nodes, vehicles, and jobs (grouped into the parameter N-V-J), as well as the time horizon, and the *edge reduction* value, which is inversely proportional to the connectivity of the graph (the higher the value, the fewer edges). Vehicles can be of type A, B, or C, and jobs come with a set of types that are eligible to execute them. For each combination of these parameters, five different problems were randomly generated. Problems belonging to the same category differ from each other in terms of tasks locations (including the additional tasks representing the depots), service time, time window (generated as a function of the time horizon), and vehicles eligible to execute them; other parameters that differ within the same category are the vehicles' operating range, the charging coefficient, and the number of vehicles available per type. Both MonoMod (see below) and the algorithms called by ComSat used Z3 4.8.9 to solve the models. All the experiments[1] were performed on an *Intel Core i7 6700K, 4.0 GHZ, 32GB RAM* running *Ubuntu-18.04 LTS*.

As mentioned in Section 2, to the best of our knowledge, the CF-EVRP presented in [16] and further developed in this work is novel. The experimental evaluation compares the monolithic model, MonoMod, presented in [16] against ComSat. Both the running time and quality of solutions are evaluated with respect to the problem parameters.

The first set of experiments compare the monolithic model (MonoMod) of the CF-EVRP presented in [16] , to ComSat on a set of relatively small problems, with up to four vehicles, seven jobs, and a time horizon of 60 time-steps; the time limit set for both methods was 1200 s. For this comparison MonoMod has been adapted to account for non-negligible service times and

[1] The implementation of the algorithm presented in Section 3 and the problem instances are available at `https://github.com/sabinoroselli/VRP.git`.

the the cost function is not included in the model, so that MonoMod returns the first feasible solution found. The choice of 1200 s is motivated by the industrial application the algorithm is designed for; while some schedules may be computed hours before they actually take place, last minute changes may happen and it is useful to know what size of problems can be solved within minutes. For MonoMod the model generation time may not be negligible; however, the comparison with ComSat is for the solving time.

Table 4 shows the results of the comparison. For smaller problems and a small time horizon, MonoMod is performing well, often outperforming ComSat, especially when the problems are infeasible. As the problems grow larger though, ComSat performs better both in terms of solving time, and in terms of number of problems solved within the time limit. As expected, a larger time horizon has a negative impact on the solving time of MonoMod, since the model is based on time discretization, and a larger T means more variables and more constraints. On the other hand, the time horizon does not seem to affect the performance of ComSat significantly; instances having the same value of N-V-J and *edge reduction*, and increasing time horizon show similar solving time. There are exceptions, but they may also be due to the different time windows, since these are generated based on the time horizon. The increase of the parameter *edge reduction* generally corresponds to an increase in the solving time for both MonoMod and ComSat, probably because having fewer edges makes it harder to find a solution if it exists, or prove infeasibility otherwise, though there are exceptions. Finally, the increase of the N-V-J parameter, as expected, corresponds to longer solving time in most cases. The reason behind the outliers, i.e. when a problem is immediately declared feasible/infeasible, is often the triviality of the problem itself. For example, when deadlines are too strict and there is no solution to the *Routing Problem* then ComSat will terminate early.

The second set of experiments evaluate the performance of ComSat on a set of larger problems, with up to eleven vehicles, fifteen jobs and a time horizon of 300 time units. Again the time limit was set to 1200 s. As for the previous set of instances, the increase in the value of N-V-J corresponds to an increase in the average solving time and a decrease in the number of solved instances per category. This time, infeasible instances are generally easier to solve, probably because of the higher number of jobs compared to the number of available vehicles (trivial infeasibility).

Overall, the evaluation showed that ComSat's performance highly depends on the problem instance; there have been rather small instances that took a long time to solve, while other relatively large instances were solved almost immediately. In general, infeasibility seems to be harder to show than feasibility. This behaviour does not come unexpected, since for ComSat, a problem cannot be declared infeasible till all solutions have been explored. Moreover, as the number of *PR* stored grows, finding a new solution becomes harder. For some problems, infeasibility may be trivial to prove, when the operating range is not large enough or the time windows are too strict, for instance. In other cases it may take several attempts before declaring a problem infeasible.

As for feasible problems, a similar reasoning applies. Sometimes it took many attempts to find a set of routes that actually led to a feasible schedule and, in general, the likelihood of finding one decreases as the number of vehicles and jobs increases. Nevertheless, even for large problems, a solution can be found rather quickly, given that enough vehicles are available.

Discussion on Optimality

So far, the focus of the experiments was on the running time required by MonoMod and ComSat to solve instances of the CF-EVRP but no on the quality of the solutions. This section focuses on the quality of the solutions produced by ComSat for a set of problem instances by comparing them to a lower bound manually computed by relaxing the capacity constraints. MonoMod has been set up to find optimal solutions by including a cost function representing the total travelled distance. To make the comparison possible, the parameters of the problem instances have been scaled down to make the problems simple enough so that the optimal solution can be found by MonoMod in reasonable time. Table 3 shows the running time and cost function value for a set of CF-EVRP instances. The problems are sorted by size, in terms of the parameters previously discussed[2]. For the instances 1 to 6, both ComSat and MonoMod have the same total travelled distance. Hence, for these instances, ComSat indeed returns the optimal solution. For the instances 7 to 9, MonoMod was not able to return a solution after 24 hours and was therefore timed out. However, the cost function value returned by ComSat matches the lower bound, hence the solutions are optimal. For the instances 1 to 3, both

[2]Details of the problem instances are available at `https://github.com/sabinoroselli/VRP.git` in the file *Optimality_test_instances.pdf*.

ComSat and MonoMod return a value higher than the lower bound, implying that the capacity constraints forced the vehicles to travel through paths longer than the shortest ones in order to serve the customers.

Table 3: Comparison of the Cost Function Value (CFV), and running time (in seconds) required to solve problem instances of the CF-EVRP. For each instance, a Lower Bound (LB) on the cost function is also provided. The time limit is set to 24 hours and "-" means that this limit was exceeded.

Instance	LB	ComSat		MonoMod	
		CFV	Time	CFV	Time
1	10	12	0.13	12	0.40
2	18	22	10.72	22	1.21
3	26	30	45.40	30	2.30
4	20	20	0.53	20	3.24
5	48	48	0.30	48	4.00
6	48	48	0.44	48	4.31
7	62	62	0.79	-	-
8	72	72	0.93	-	-
9	76	76	0.95	-	-
10	10	12	0.15	10	0.19

However, ComSat is not guaranteed to find the optimal solution, as shown by instance 10 above. This is due to the way the sub-problems are structured. For a given set of routes ComSat will try to find a feasible set of paths that satisfies the capacity constraints. If such set of paths exists, ComSat terminates with a feasible solution. Nevertheless, there could exist another set of routes for which there exists a set of paths that are cumulatively shorter and satisfies the capacity constraints. This is clarified with the following example, depcited in Fig. 4.

$\mathcal{N} = \{1, \ldots, 7\},\ \mathcal{N}H = \varnothing, \mathcal{O} = \{1, 6\}$

$\mathcal{E} = \{(1,2), (2,3), (2,5), (3,4), (4,7), (5,6), (6,7)\}$

$\mathcal{J} = \{j1, j2\},\ \mathcal{K} = \{i1, i2 \mid \forall i \in \mathcal{J}\}$

$L_{j11} = 5,\ L_{j21} = 2,\ L_{j31} = 4$

$\mathcal{P}_{j11} = \varnothing, \mathcal{P}_{j21} = \varnothing, \mathcal{P}_{j31} = \varnothing$

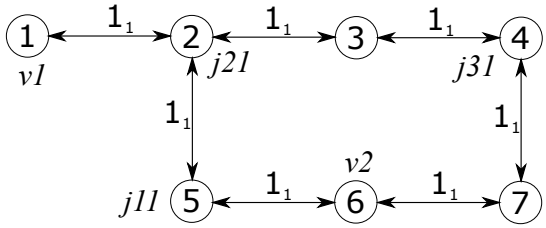

Figure 4: Finite, strongly connected, weighted, directed graph representing the plant layout for a problem instance of the CF-EVRP that cannot be solved to optimality by ComSat.

$l_{j11} = 2,\ l_{j21} = 2,\ l_{j31} = 2$

$u_{j11} = 2,\ u_{j21} = 5,\ u_{j31} = 7$

$S_{j11} = 2,\ S_{j21} = 1, S_{j31} = 1$

$\mathcal{V} = \{v1, v2\},\ \mathcal{V}_{j1} = \{v1\},\ \mathcal{V}_{j2} = \{v2\},\ \mathcal{V}_{j3} = \{v2\}$

$OR = 10, C = 1,\ D = 1,\ \rho = 1,\ v = 1,\ T = 13$

Vehicle $v1$ needs to travel from node 1 to node 5 in order to execute task $j11$; vehicle $v2$ will be assigned to both task $j21$ and $j31$, respectively located at nodes 2 and 4. Since these tasks are equidistant from $v2$'s location (node 6), serving one before the other or the other way around would result in the same cost for a route, hence ComSat may choose one as well as the other. Assuming that task $j21$ is served before $j31$, there will be a conflict; in fact, vehicle $v1$ occupies node 5 to serve task $j11$, due to its time window and service time. If ComSat had come to such a situation, it would call the *PathChanger* function. There is no different path for $v1$ that would be feasible against the time window of task $j11$; however, $v2$ could reach node 2 by passing through nodes 7, 4, and 3 and still meet the time window of task $j21$. It would then go back to node 4 and execute task $j31$. The total length of such a route will be 8. However, serving task $j31$ before $j21$ would not result in a conflict with $v1$, so there would be no need to look for alternative paths; the route length in this case would only be 6. Thus, as long as a solution can be found without needing to change paths, ComSat will return an optimal solution, else optimality is not guaranteed.

7 Conclusions

This paper presents the compositional algorithm ComSat to solve the CF-EVRP. It is proven that the algorithm is sound and complete if there are no cycles in the paths. ComSat was compared to the performance of a monolithic model for the CF-EVRP, which showed that as the problems' size grows ComSat outperforms the monolithic model. ComSat's performance was also evaluated over a set of larger generated problem instances, which showed that it can solve problems counting up to 11 vehicles and 15 jobs in a reasonably short time. Experimental data shows that the solving time for similar-sized problems can be very different.

For problems whose feasibility is not straightforward to achieve, it is an open research question on how to avoid a large number of iterations but at the same time not increase the sub-problems complexity. The compositional structure of ComSat allows to replace the algorithms that solve the sub-problems without affecting the overall soundness and completeness, as long as the requirements in Section 3 are met. Experiments show the *Router* and the *PathsChanger* to be bottlenecks when a problem instance is hard to solve. Therefore future work will focus on improving these algorithms. As for the *Router*, there exist several algorithms to solve VRPs that could be adapted to solve the *Routing Problem* described in Section 4.2; as for the *PathsChanger*, extraction of *Unsat-Core* when the *Capacity Verification Problem* is infeasible could provide useful information to guide the search for alternative paths.

However, for industrial scenarios, feasibility is typically straightforward to achieve, due to a sufficient number of available vehicles and sufficient capacity of the road segments. In such scenarios, ComSat can find feasible solutions with few iterations, being therefore useful for many industrial applications.

References

[1] K. Azadeh, M. deKoster, and D. Roy, "Robotized warehouse systems: Developments and research opportunities," *ERIM Report Series Research in Management*, no. ERS-2017-009-LIS, 2017.

[2] H. Zhang, H. Ge, J. Yang, and Y. Tong, "Review of vehicle routing problems: Models, classification and solving algorithms," *Archives of Computational Methods in Engineering*, pp. 1–27, 2021.

Table 4: Comparison of ComSat and the monolithic model (MonoMod) for the CF-EVRP over a set of generated problem instances. Instances are sorted by the parameters N-V-J (nodes, vehicles, jobs), value of edge reduction, and time horizon. For each resulting class, five instances are evaluated and the number of feasible (Feas) and unfeasible (Unfeas) ones is reported, together with the average solving time (in seconds) for that specific class. When the number of instances does not add up to five it is because the running time exceeded the time limit of 1200 seconds. The symbol "-" means that no instance for that category was either feasible or unfeasible, depending on where the symbol appears.

		N-V-J																							
		15-3-5 Edge Reduction												25-4-7 Edge Reduction											
T		0				25				50				0				25				50			
		Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas
20	ComSat	1.74	1	120.43	4	0.63	1	106.75	4	-	0	49.12	5	-	0	539.04	4	-	0	367.59	4	-	0	284.82	4
	MonoMod	5.7	1	5.81	4	11.87	1	8.74	4	-	0	12.13	5	-	0	21.35	5	-	0	19.98	5	-	0	15.43	5
25	ComSat	2.71	2	125.16	3	0.67	2	90.28	3	1.23	1	51.83	4	6.27	1	599.9	3	-	0	315.15	4	-	0	190.63	4
	MonoMod	10.51	3	235.28	2	16.08	3	13.52	1	26.63	1	217.15	3	100.16	1	63.24	4	-	0	44.34	5	-	0	39.35	5
30	ComSat	6.72	3	0.36	2	3.73	2	179.19	3	2.78	1	89.03	4	15.3	2	804.06	2	4.19	1	410.97	3	98.25	1	200.49	3
	MonoMod	25.71	3	4.45	1	80.33	3	9.14	1	112.24	1	57.64	2	292.1	2	145.42	2	242.48	2	70.34	2	159.43	1	242.91	3
40	ComSat	1.75	3	0.37	2	6.16	3	0.37	2	3.83	3	0.37	2	5.23	4	671.25	1	192.15	4	462.68	1	4.98	3	190.19	1
	MonoMod	342.44	3	9.41	1	162.11	2	34.59	1	210.6	2	19.71	1	403.96	2	-	0	534.69	2	-	0	414.29	2	-	0
50	ComSat	0.74	3	0.39	2	4.99	3	0.37	2	2.04	3	0.36	2	5.08	4	964.66	1	7.66	4	466.4	1	14.54	3	345.57	2
	MonoMod	237.64	3	16.14	1	412.25	3	15.89	1	704.12	3	20.49	1	497.04	1	-	0	628.88	1	-	0	-	0	-	0
60	ComSat	1.58	3	0.36	2	6.46	3	0.37	2	8.01	3	0.36	2	4.29	5	-	0	15.02	5	-	0	5.2	5	-	0
	MonoMod	462.53	2	-	0	553.51	2	-	0	742.8	2	-	0	-	0	-	0	-	0	-	0	-	0	-	0

Table 5: Evaluation of ComSat for the CF-EVRP over a set of generated problem instances. Instances are sorted by the parameters N-V-J (nodes, vehicles, jobs), and value of edge reduction in the columns, and time horizon in the rows. For each resulting class, five instances are evaluated and the number of feasible (Feas) and unfeasible (Unfeas) ones is reported, together with the average solving time (in seconds) for that specific class. When the number of instances does not add up to five it is because the running time exceeded the time limit of 1200 seconds. The symbol "-" means that no instance for that category was either feasible or unfeasible, depending on where the symbol appears.

	N-V-J																							
	35-6-8 Edge Reduction												35-07-10 Edge Reduction											
T	0				25				50				0				25				50			
	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas
40	98.32	3	292.11	2	42.93	2	383.28	3	10.86	2	351.63	3	5.49	1	1.33	2	-	0	1.33	2	-	0	403.6	3
70	79.55	3	269.74	2	19.89	2	0.86	3	42.98	2	0.86	3	238.65	3	1.32	2	127.31	1	1.33	2	4.59	1	1.31	2
100	15.35	3	89.12	2	79.9	2	281.04	3	73.86	2	281.66	3	452.1	3	1.31	2	22.01	1	1.33	2	7.48	1	1.31	2
150	8.94	3	100.53	2	47.33	3	0.87	2	56.84	2	0.86	3	28.44	3	1.3	2	16.79	1	1.39	2	32.41	1	1.31	2
200	6.48	4	0.86	1	23.19	2	351.15	3	102.8	2	350.46	3	202.68	3	1.32	2	66.64	1	1.34	2	21.18	1	1.29	2
300	8.99	4	0.88	1	34.1	3	0.86	2	17.21	2	0.86	3	74.12	3	1.34	2	6.36	1	1.33	2	79.27	1	1.35	2

	N-V-J																							
	35-9-12 Edge Reduction												35-11-15 Edge Reduction											
T	0				25				50				0				25				50			
	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas	Av.(sec)	Feas	Av.(sec)	Unfeas
40	486.31	2	1.6	3	311.95	1	1.88	2	-	0	1.9	3	-	0	3.17	1	-	0	3.19	4	-	0	3.24	4
70	221.11	3	1.89	1	107.44	1	321.05	2	143.61	1	1.94	3	-	0	3.26	1	-	0	3.33	4	-	0	3.32	4
100	77.16	2	1.88	2	28.02	1	1.9	2	111.09	1	1.92	3	437.57	2	-	0	341.98	1	3.17	3	-	0	3.2	3
150	224.31	3	1.92	2	255.06	2	1.95	2	311.23	2	1.99	3	337.26	2	-	0	-	0	3.22	3	-	0	3.26	3
200	489.57	3	1.89	2	27.59	1	1.88	2	178.64	1	1.94	3	18.38	1	3.19	2	788.91	1	3.27	4	661.28	1	3.22	4
300	238.88	3	1.92	2	391.58	1	1.92	2	349.04	1	1.94	3	646.29	2	-	0	774.71	1	3.26	3	26.26	1	3.31	3

[3] N. Brahimi and T. Aouam, "Multi-item production routing problem with backordering: A MILP approach," *International Journal of Production Research*, vol. 54, no. 4, pp. 1076–1093, 2016.

[4] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, part I: Route construction and local search algorithms," *Transportation science*, vol. 39, no. 1, pp. 104–118, 2005.

[5] S. Riazi, K. Bengtsson, and B. Lennartson, "Energy optimization of large-scale AGV systems," *IEEE Transactions on automation science and engineering*, vol. 18, no. 2, pp. 638–649, 2020.

[6] B. M. Baker and M. Ayechew, "A genetic algorithm for the vehicle routing problem," *Computers & Operations Research*, vol. 30, no. 5, pp. 787–800, 2003.

[7] Y.-J. Gong, J. Zhang, O. Liu, R.-Z. Huang, H. S.-H. Chung, and Y.-H. Shi, "Optimizing the vehicle routing problem with time windows: A discrete particle swarm optimization approach," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 2, pp. 254–267, 2011.

[8] F. Rossi, R. Iglesias, M. Alizadeh, and M. Pavone, "On the interaction between autonomous mobility-on-demand systems and the power network: Models and coordination algorithms," *IEEE Transactions on Control of Network Systems*, vol. 7, no. 1, pp. 384–397, 2019.

[9] E. Thanos, T. Wauters, and G. Vanden Berghe, "Dispatch and conflict-free routing of capacitated vehicles with storage stack allocation," *Journal of the Operational Research Society*, pp. 1–14, 2019.

[10] E. Jernheden, C. Lindström, R. Persson, *et al.*, "Comparison of exact and approximate methods for the vehicle routing problem with time windows," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2020, pp. 378–383.

[11] M. Desrochers, J. Desrosiers, and M. Solomon, "A new optimization algorithm for the vehicle routing problem with time windows," *Operations research*, vol. 40, no. 2, pp. 342–354, 1992.

[12] S. F. Roselli, K. Bengtsson, and K. Åkesson, "SMT solvers for job-shop scheduling problems: Models comparison and performance evaluation," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2018, pp. 547–552.

[13] ——, "SMT solvers for flexible job-shop scheduling problems: A computational analysis," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2019.

[14] C. Barrett and C. Tinelli, "Satisfiability modulo theories," *Handbook of model checking*, pp. 305–343, 2018.

[15] L. De Moura and N. Bjørner, "Satisfiability modulo theories: Introduction and applications," *Commun. ACM*, vol. 54, no. 9, pp. 69–77, Sep. 2011, ISSN: 0001-0782.

[16] S. Roselli, M. Fabian, and K. Åkesson, "Solving the Electric-Conflict Free-Vehicle Routing Problem Using SMT Solvers," *2021 29th Mediterranean Conference on Control and Automation (MED)*, pp. 542–547, 2021.

[17] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νZ-an optimizing SMT solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2015, pp. 194–199.

[18] S. Roselli, M. Fabian, and K. Åkesson, "An SMT Based Compositional Algorithm to Solve a Conflict-Free Electric Vehicle Routing Problem," *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pp. 1364–1369, 2021.

[19] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[20] N. Smolic-Rocak, S. Bogdan, Z. Kovacic, and T. Petrovic, "Time windows based dynamic routing in multi-agv systems," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 1, pp. 151–155, 2009.

[21] A. Lim and F. Wang, "Multi-depot vehicle routing problem: A one-stage approach," *IEEE transactions on Automation Science and Engineering*, vol. 2, no. 4, pp. 397–402, 2005.

[22] H. C. Lau, T. Chan, W. Tsui, and W. Pang, "Application of genetic algorithms to solve the multidepot vehicle routing problem," *IEEE transactions on automation science and engineering*, vol. 7, no. 2, pp. 383–392, 2009.

[23] N. N. Krishnamurthy, R. Batta, and M. H. Karwan, "Developing conflict-free routes for automated guided vehicles," *Operations Research*, vol. 41, no. 6, pp. 1077–1090, 1993.

[24] A. I. Corréa, A. Langevin, and L.-M. Rousseau, "Scheduling and routing of automated guided vehicles: A hybrid approach," *Computers & operations research*, vol. 34, no. 6, pp. 1688–1707, 2007.

[25] J. F. Bard, L. Huang, M. Dror, and P. Jaillet, "A branch and cut algorithm for the VRP with satellite facilities," *IIE Transactions*, vol. 30, no. 9, pp. 821–834, 1998.

[26] M. Schneider, A. Stenger, and D. Goeke, "The electric vehicle-routing problem with time windows and recharging stations," *Transportation Science*, vol. 48, no. 4, pp. 500–520, 2014.

[27] T. Nishi and R. Maeno, "Petri net decomposition approach to optimization of route planning problems for AGV systems," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 3, pp. 523–537, 2010.

[28] M. Mousavi, H. J. Yap, S. N. Musa, F. Tahriri, and S. Z. Md Dawal, "Multi-objective AGV scheduling in an FMS using a hybrid of genetic algorithm and particle swarm optimization," *PloS one*, vol. 12, no. 3, e0169817, 2017.

[29] Y. Liu, S. Ji, Z. Su, and D. Guo, "Multi-objective AGV scheduling in an automatic sorting system of an unmanned (intelligent) warehouse by using two adaptive genetic algorithms and a multi-adaptive genetic algorithm," *PloS one*, vol. 14, no. 12, 2019.

[30] H. F. Rahman, M. N. Janardhanan, and P. Nielsen, "An integrated approach for line balancing and AGV scheduling towards smart assembly systems," *Assembly Automation*, 2020.

[31] N. Singh, Q.-V. Dang, A. Akcay, I. Adan, and T. Martagan, "A matheuristic for AGV scheduling with battery constraints," *European Journal of Operational Research*, vol. 298, no. 3, pp. 855–873, 2022, ISSN: 0377-2217.

[32] M. Saidi-Mehrabad, S. Dehnavi-Arani, F. Evazabadian, and V. Mahmoodian, "An ant colony algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs," *Computers & Industrial Engineering*, vol. 86, pp. 2–13, 2015.

[33] R. Yuan, T. Dong, and J. Li, "Research on the collision-free path planning of multi-AGVs system based on improved A* algorithm," *American Journal of Operations Research*, vol. 6, no. 6, pp. 442–449, 2016.

[34] K. Murakami, "Time-space network model and MILP formulation of the conflict-free routing problem of a capacitated AGV system," *Computers & Industrial Engineering*, 2020.

[35] M. Zhong, Y. Yang, Y. Dessouky, and O. Postolache, "Multi-AGV scheduling for conflict-free path planning in automated container terminals," *Computers & Industrial Engineering*, vol. 142, 2020.

[36] L. W. Rizkallah, M. F. Ahmed, and N. M. Darwish, "SMT-LH: A new satisfiability modulo theory-based technique for solving vehicle routing problem with time window constraints," *The Computer Journal*, vol. 63, no. 1, pp. 91–104, 2020.

[37] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[38] C. Sinz, "Towards an optimal CNF encoding of boolean cardinality constraints," in *International conference on principles and practice of constraint programming*, Springer, 2005, pp. 827–831.

[39] F. A. Aloul, B. Al Rawi, and M. Aboelaze, "Identifying the shortest path in large networks using boolean satisfiability," in *2006 3rd International Conference on Electrical and Electronics Engineering*, IEEE, 2006, pp. 1–4.

[40] C. Tinelli and M. Harandi, "A new correctness proof of the nelson-oppen combination procedure," in *Frontiers of Combining Systems*, Springer, 1996, pp. 103–119.

PAPER E

Leveraging Conflicting Constraints in Solving Vehicle Routing Problems

Sabino Francesco Roselli, Remco Vader, Martin Fabian, Knut Åkesson

Proceedings of 16th IFAC Workshop on Discrete Event Systems (WODES)
September 7-9, 2022 - Prague, Czechia

The layout has been revised.

Abstract

The Conflict-Free Electric Vehicle Routing Problem (CF-EVRP) is a combinatorial optimization problem of designing routes for vehicles to visit customers such that a cost function, typically the number of vehicles or the total travelled distance, is minimized. The CF-EVRP involves constraints such as time windows on the delivery to the customers, limited operating range of the vehicles, and limited capacity on the number of vehicles that a road segment can simultaneously accommodate. In previous work, the compositional algorithm *ComSat* was introduced and that solves the CF-EVRP by breaking it down into sub-problems and iteratively solve them to build an overall solution. Though ComSat showed good performance in general, some problems took significant time to solve due to the high number of iterations required to find solutions that satisfy the road segments' capacity constraints. The bottleneck is the *Path Changing Problem*, i.e., the sub-problem of finding a new set of shortest paths to connect a subset of the customers, disregarding previously found shortest paths. This paper presents an improved version of the *PathsChanger* function to solve the *Path Changing Problem* that exploits the *unsatisfiable core*, i.e., information on which constraints conflict, to guide the search for feasible solutions. Experiments show faster convergence to feasible solutions compared to the previous version of *PathsChanger*.

1 Introduction

We consider scheduling a fleet of mobile robots, in the sequel referred to as Automated Guided Vehicles (AGVs), that pick-up and deliver components to workstations within specified time-windows. The AGVs move on a predefined road network, where each road segment has a maximum number of AGVs it can accommodate at a specific time. The problem is motivated by an industrial need to develop more flexible logistic systems to deliver components

just-in-time to an assembly line.

In this scenario, in addition to time-windows in which the components should be delivered, a scheduler needs to consider additional constraints. First, AGVs have a limited operating range and need to recharge their battery when the state-of-charge becomes low. Second, jobs have specific requirements on the AGV eligible to execute them. Finally, the number of AGVs on road segments and workstations are limited to allow low-level trajectory planning problems to be feasible. Thus, we define the *capacity* of the road segments, intersections, and workstations and include *capacity constraints*. A schedule is said to be *conflict-free* if it fulfills the capacity constraints at all times.

The problem of computing conflict-free routes was first introduced in [1] and tackled by means of column generation. In [2], conflict-free routing in combination with scheduling of jobs for flexible manufacturing systems is discussed. An ant colony algorithm is applied to the problem of job shop scheduling and conflict free routing of AGVs by [3]. In [4], a collision-free path planning for multi AGV systems based on the A^* algorithm is presented. Another heuristic approach to solve the conflict-free routing problem with storage allocation is presented by [5]. In [6], a MILP formulation to design conflict-free routes for capacitated vehicles is presented. In [7] is presented a hybrid evolutionary algorithm to deal with conflict-free AGV scheduling in automated container terminals, and [8] handles the problem of conflict-free routing of AGVs by a meta-heuristic improvement strategy based on large neighbourhood search. Hence, conflict-free routing and scheduling has been addressed previously, but to the best of our knowledge, there is no work in the literature that tackles all above mentioned constraints at once. Therefore, [9] introduced the *Conflict-Free Electric Vehicle Routing Problem* (CF-EVRP). The CF-EVRP is an extension of the vehicle routing problem (VRP) [10], involving the additional constraints. In [11] a compositional algorithm, ComSat, for solving the CF-EVRP is proposed. ComSat breaks down CF-EVRP into sub-problems and iteratively solves these to find a feasible solution to the overall problem. Experimental and analytical evaluation shows that ComSat generates high-quality but not necessarily optimal solutions. Briefly, ComSat computes routes to serve the customers, and assigns vehicles to the routes attempting to make the execution of the system conflict-free. In a plant there can be several ways to travel from one customer's location to another. Initially, ComSat uses the shortest paths among the customers' locations when

designing the routes. However, if a feasible schedule cannot be achieved using the shortest paths, alternative paths have to be found, which is handled by the *Conflict-free Paths Search* (CFPS). CFPS is composed of two main functions; the *PathsChanger* function, that finds alternative sets of paths if the current schedule violates the capacity constraints, and the *CapacityVerifier* function, that checks whether the schedule is conflict-free or not.

Experiments show that when a solution computed using the shortest paths violates the capacity constraints, finding alternative paths using the *PathsChanger* function may require multiple iterations. This does not come unexpected, since the number of possible paths in a graph can be high, and minimizing the cumulative length while looking for alternative paths does not guarantee that the schedule will be conflict-free. In this paper we focus on the CFPS and present improved versions of the *PathsChanger* and *CapacityVerifier* that, in many cases, find feasible solutions faster.

The sub-problems in ComSat are modelled as Satisfiability Modulo Theory (SMT) problems [12], [13], as SMT solvers have shown to be efficient in solving combinatorial problems [14].

Moreover, some SMT solvers come with algorithms that allow them to deal with optimization problems [15]. Two sub-problems in ComSat, marked by the round boxes in Fig. 1 (see below) are optimization problems.

For the CFPS polynomial time algorithms exist to find paths in graphs, [16]. However, modelling the *Path Changing Problem* as an SMT problem is beneficial as it allows to define problem-specific requirements, such as not returning solutions that are already proven infeasible because they violate the capacity constraints. Moreover, when a problem is infeasible, SMT solvers have the ability to return a *Minimal Unsatisfiable Core* (*MUC*) [17], i.e., one of the (possibly many) smallest subsets of constraints that make the problem infeasible. The *MUC* can provide useful information about why a problem is infeasible and can therefore be used to guide the search towards a feasible solution [18].

When dealing with the CF-EVRP, the *MUC* can be extracted when the *Capacity Verification Problem* is infeasible and used to define additional constraints for the *Path Changing Problem*, to increase the chances of finding a feasible schedule.

The contributions in this paper are: (i) exploitation of SMT solvers' *MUC* to extract information about the infeasibility of an SMT formula representing

a conflicting schedule for a VRP; (ii) use of such information to find conflict-free schedules; (iii) performance comparison between the unguided and *MUC* guided paths search over a set of CF-EVRP problem instances.

The remainder of the paper is organized as follows. Preliminaries are presented in Section 2. Section 3 presents the mathematical models of the sub-problems that form the CFPS and how it is improved using the *MUC* from the *Capacity Verification Problem*. Proof of soundness and completeness of the procedure is provided in Section 4. In Section 5, the results of the analysis over a set of problem instances are presented. Finally, conclusions are drawn in Section 6.

2 Preliminaries

In the CF-EVRP the plant layout is represented by a finite, strongly connected, weighted, directed graph, where edges represent road segments and nodes represent either intersections between road segments or customers' locations. A customer is defined by a unique (numerical) identifier, a location, and a time window, i.e., a lower and upper bound that represent the earliest and latest arrival time allowed to serve the customer. Edges have two attributes, the first representing the road segment's length, and the second its capacity. The capacity is 2 if two vehicles can simultaneously travel in opposite directions, 1 otherwise.

The following definitions are provided:

- *Node*: a location in the plant. A node can only accommodate one vehicle at a time unless it is a *hub* node that can accommodate an arbitrary number of vehicles.

  $\mathcal{N}$: a finite set of nodes.

  $\mathcal{N}_H \subseteq \mathcal{N}$: the set of hub nodes.

- *Edge*: a road segment that connects two nodes.

  $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$: the finite set of direct edges.

  $\bar{e}$: the reverse edge of edge $e \in \mathcal{E}$.

  $d_e \in \mathbb{R}_+$: the length of edge $e \in \mathcal{E}$.

  $g_e \in \{1, 2\}$: the capacity of edge $e \in \mathcal{E}$.

- *Time horizon*: a fixed, continuous point of time when all jobs have ended, assuming they start at time 0.

  T: the time horizon.

- *Customer*: Entity representing a task to be executed by a vehicle, e.g., a pickup or delivery of material, that needs to be visited exactly once by the vehicle. A customer is always associated with a node where the pickup/delivery operation is executed, and has a time window indicating the earliest and latest time at which it can be visited. Unless explicitly given, the time window is the entire time span $[0, T]$.

  Let $\mathcal{K}$ be the finite set of all customers, and let

  $l_k, u_k \in \mathbb{R}_+,\ k \in \mathcal{K}$ be the time window's lower (l_k) and upper (u_k) bound for customer k such that $u_k > l_k$.

  Also let $s_k \in \mathbb{R}_+$ and $L_k \in \mathcal{N}$, for $k \in \mathcal{K}$, be the service time and location of customer k, respectively.

- *Route*: an ordered set of unique customers.

  $$r_j = \langle k_{j1}, \ldots, k_{jm} \rangle,\ m \leq |\mathcal{K}|,\ k_{ji} \in \mathcal{K},$$
  $$i = 1, \ldots, m,\ k_{jl} \neq k_{ji} \text{ for } i \neq l.$$

  A route can at most include all customers, therefore $m \leq |\mathcal{K}|$.

- *Route set*: a set of routes such that each customer belongs to exactly one route, thus guaranteeing that all customers are served.

  $$\mathcal{R} = \{r_1, \ldots, r_m\},\ m \leq |\mathcal{K}|$$

  A route contains at least one customer, hence $m \leq |\mathcal{K}|$.

- *Route start*: the starting time τ_r of route r, computed by the function *Assign*. Γ is the set that contains the *route start* of each route.

  $$\Gamma = \{\tau_r \in \mathbb{R} \,|\, r \in \mathcal{R}\}$$

- *Pair Set of route r*: set containing the sequence of customers of a route $r = \langle k_1, \ldots, k_m \rangle$, grouped as pairs in sequence.

  $$\mathcal{P}_r = \{\langle k_1, k_2 \rangle, \langle k_2, k_3 \rangle, \ldots, \langle k_{m-1}, k_m \rangle\}$$

- *Path*: ordered set of unique nodes. It is used to keep track of how vehicles are travelling among customers of routes, since each pair of customers in a route is connected by a path.

$$\theta_p = \langle n_1, \ldots, n_m \rangle,\ p \in \mathcal{P}_r,\ m \leq |\mathcal{N}|,$$
$$n_i \in \mathcal{N},\ i = 1, \ldots, m$$

- *Edge sequence*: ordered set of unique edges for a given path θ_p.

$$\delta_p = \langle e_1, \ldots, e_m \rangle,\ p \in \mathcal{P}_r,\ m = |\theta_p| - 1,$$
$$e_i \in \mathcal{E},\ i = 1, \ldots, m$$

In order to clarify which part of ComSat is analyzed and improved in this work, let us recap briefly how the algorithm works. Fig. 1 shows a simplified flowchart of ComSat that illustrate the concepts of this paper. The first step of ComSat is to design a set of routes $\mathcal{R}$ to serve all the customers; at this point, the shortest path between any two customers is computed using Dijkstra's algorithm [19].

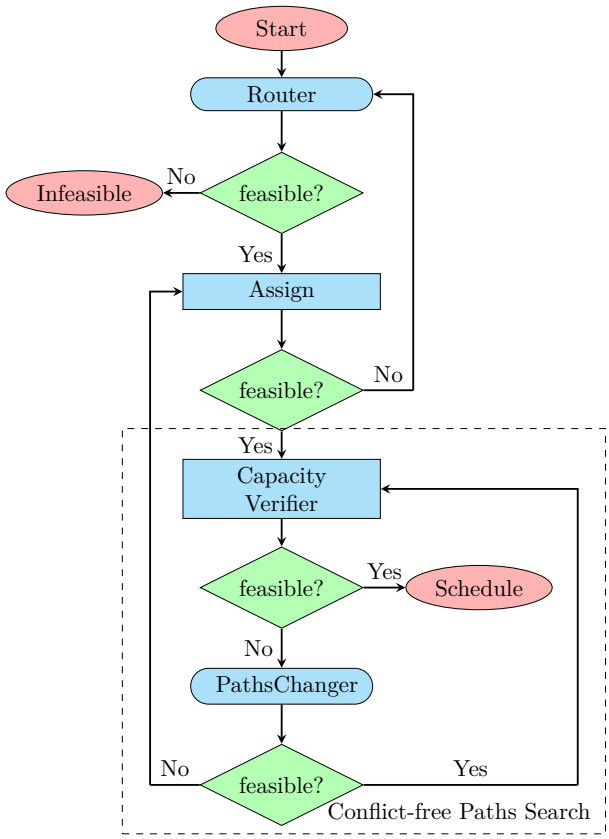

Figure 1: Flowchart of ComSat.

This optimization problem is handled by the function *Router* and must guarantee that the routes meet specific requirements such as maximum length, specific ordering among the customers and time windows. If this step is infeasible the CF-EVRP instance has no solution and the algorithm terminates. If this step is feasible, the function *Assign* will try to allocate available vehicles to the routes and compute a start time τ_r, $\forall r \in \mathcal{R}$, to the routes. If this step is infeasible then *Router* will try to find different routes, but if it is feasible, the *CapacityVerifier* checks if the current set of routes is conflict-free. More details on the functions *Router* and *Assign* can be found in [11].

2.1 *The minimal Unsat Core*

For infeasible problems, there can be identified a subset of the constraints that conflict, meaning they cannot all simultaneously be satisified. Such a subset is called an *Unsat Core*. An *Unsat Core* with the property that removing any one of the constraints makes the *Unsat Core* feasible, is said to be *minimal*.

Formally, given an SMT formula φ and set of conflicting constraints $\mathcal{C} \subseteq \varphi$, $\mathcal{C}$ is a *MUC* of φ if removing any constraint $\mathcal{C}_i \in \mathcal{C}$ makes $\mathcal{C} \setminus \mathcal{C}_i$ no longer infeasible; removing $\mathcal{C}$ removes the particular conflict represented by the *MUC*. Consequently, for an infeasible problem with a *MUC* $\mathcal{C}$, adding to the problem a constraint that prevents all the constraints in $\mathcal{C}$ to be simultaneously active will resolve this particular conflict.

The naïve approach to *MUC* extraction, [20], successively removes constraints and solves the problem again; if the problem is still infeasible after a constraint has been removed that constraint does not belong to a *MUC*. There exist more efficient approaches though; the *MUC* [21] algorithm based on efficient manipulation of *Binary Decision Trees* guarantees the extraction of a *minimal Unsat Core*. [22] presents an algorithm based on the resolution graph [23] for *MUC* extraction. [24] improves the resolution based algorithm using *model rotation* and *path strengthening*.

3 The Conflict-free Paths Search

In this section the two sub-problems that form the CFPS are presented. The *Capacity Verification Problem* is modelled as a job shop problem (JSP), in order to exploit the good performance of the SMT solver Z3 [25] in dealing with JSPs, as demonstrated in [26]. The model formulation for the *Path Changing Problem* is inspired by [27].

The following logical operators are used as a shorthand to express cardinality constraints [28] in the sub-problems:

$\mathrm{EN}(A, n)$: exactly n variables in the set A are true;

$\mathrm{If}(c, o_1, o_2)$: if c is *true* returns o_1, else returns o_2.

We will write $\mathrm{EN}_{m \in M}(m, n)$ to denote $\mathrm{EN}(\bigcup_{m \in M} \{m\}, n)$ in order to shorten the notation.

3.1 *The Capacity Verification Problem*

The *Capacity Verification Problem* aims to find a feasible schedule for the vehicles, where the routes that the vehicles are assigned to satisfy the capacity constraints of the edges.

In this work the *Capacity Verification Problem*, as defined in [11], has been extended to account for pairs as well, since the information about conflicts must be related to a specific pair to define additional constraints in the *PathsChanger*.

Let n_{rpe} be the node visited before edge e of pair p of route r, and let e_{rpn} be the node visited before node n on pair p of route r. Similarly, let n^{rpe} be the node visited after edge e of pair p of route r, and let e^{rpn} be the edge visited after node n on pair p route r. Let p_r^0 be the first pair of route r and n_r^* be its starting node.

Example of Routes, Pairs, Nodes, and Edges

Let $\mathcal{K} = \{k_1, \ldots, k_7\}$, $\mathcal{N} = \{n_1, \ldots, n_{20}\}$, and $L_{k_1} = n_1$ and $L_{k_2} = n_7$, and assume two routes designed to serve all customers: $r_1 = \langle k_1, k_2, k_5, k_7 \rangle$, $r_2 = \langle k_3, k_4, k_6 \rangle$. In order to clarify the notation introduced above, let us analyze r_1. First, the set of pairs for r_1 is defined as $\mathcal{P}_{r_1} = \{\langle k_1, k_2 \rangle, \langle k_2, k_5 \rangle, \langle k_5, k_7 \rangle\}$. Then, let us assume that the *path* and *edge sequence* for pair $\langle k_1, k_2 \rangle$ are the following:

$\theta_{\langle k_1, k_2 \rangle} = \langle n_1, n_2, n_4, n_5, n_7 \rangle$,
$\delta_{\langle k_1, k_2 \rangle} = \langle \langle n_1, n_2 \rangle, \langle n_2, n_4 \rangle, \langle n_4, n_5 \rangle, \langle n_5, n_7 \rangle \rangle$.

Then $p_{r_1}^0 = \langle k_1, k_2 \rangle$ and $n_{r_1}^* = n_1$. Also, let $p = \langle k_1, k_2 \rangle$; then for $e = \langle n_1, n_2 \rangle$, $n_{r_1 pe} = n_1$, and $n^{r_1 pe} = n_2$; for $n = n_1$, $e^{r_1 pn} = \langle n_1, n_2 \rangle$, and for $n = n_2$, $e_{r_1 pn} = \langle n_1, n_2 \rangle$.

For each node it must also be specified whether there exists a time window, since some of the nodes are only intersections of road segments in the real plant, while others are actual customers. Let l_{rpn} and u_{rpn} be the earliest and latest arrival time, respectively, at node n of pair p of route r; let s_{rpn} be the service time at node n of pair p of route r. Finally, let $\gamma > 0$ be a small real constant used to prevent *swapping* of vehicles' positions between a node and the previous or following edge.

The *Capacity Verification Problem* decision variables are:

x_{rpn}: non-negative real variable that models when a vehicle executing route r starts using node n in pair p;

y_{rpe}: non-negative real variable that models when a vehicle executing route r starts using edge e in pair p;

The model for the *Capacity Verification Problem* is:

$$x_{rp_r^0 n_r^*} \geq \tau_r,\ \forall r \in \mathcal{R} \tag{E.1}$$

$$y_{rpe} \geq x_{rpn_{rpe}} + s_{rpn_{rpe}},\ \forall r \in \mathcal{R},\ p \in \mathcal{P}_r,\ e \in \delta_p \tag{E.2}$$

$$x_{rpn} = y_{rpe_{rpn}} + d_{e_{rpn}},\ \forall r \in \mathcal{R},\ p \in \mathcal{P}_r,\ n \in \theta_p \tag{E.3}$$

$$\begin{aligned} &x_{rpn} \geq l_{rpn} \wedge x_{rpn} \leq u_{rpn}, \\ &\quad \forall r \in \mathcal{R},\ p \in \mathcal{P}_r,\ n \in \theta_p \end{aligned} \tag{E.4}$$

$$\begin{aligned} &x_{r_1 p_1 n} \geq y_{r_2 p_2 e^{r_1 p_1 n}} + \gamma \ \vee \ x_{r_2 p_2 n} \geq y_{r_1 p_1 e^{r_2 p_2 n}} + \gamma, \\ &\quad \forall r_1, r_2 \in \mathcal{R},\ r_1 \neq r_2,\ p_1 \in \mathcal{P}_{r_1},\ p_2 \in \mathcal{P}_{r_2} \\ &\quad n \in \theta_{p1} \cap \theta_{p2},\ n \notin \mathcal{N}_H \end{aligned} \tag{E.5}$$

$$\begin{aligned} &y_{r_1 p_1 e} \geq y_{r_2 p_2 e} + \gamma \vee y_{r_2 p_2 e} \geq y_{r_1 p_1 e} + \gamma, \\ &\quad \forall r_1, r_2 \in \mathcal{R},\ r_1 \neq r_2,\ p_1 \in \mathcal{P}_{r_1},\ p_2 \in \mathcal{P}_{r_2}, \\ &\quad e \in \delta_{p_1} \cap \delta_{p_2} \end{aligned} \tag{E.6}$$

$$\begin{aligned} &y_{r_1 p_1 e_1} \geq y_{r_2 p_2 e_2} + d_{e_2} \ \vee y_{r_2 p_2 e_2} \geq y_{r_1 p_2 e_1} + d_{e_1}, \\ &\quad \forall r_1, r_2 \in \mathcal{R},\ r_1 \neq r_2,\ p_1 \in \mathcal{P}_{r_1},\ p_2 \in \mathcal{P}_{r_2}, \\ &\quad e_1 \in \delta_{p_1},\ e_2 \in \delta_{p_2},\ e_1 = \bar{e}_2,\ g_{e_1} = g_{e_2} = 1 \end{aligned} \tag{E.7}$$

(E.1) constrains the start time of a route; (E.2) and (E.3) define the precedence among nodes and edges to visit in a route; (E.4) enforces time windows on the nodes that correspond to the customers; (E.5) prevents vehicles from using the same node at the same time; (E.6) and (E.7) constrain the transit of vehicles over the same edge. If two vehicles are using the same edge from the same node, one has to start at least γ after the other and if two vehicles are using the same edge from opposite nodes, one has to fully transit before the other one can start.

Based on the model described above, the algorithm *CapacityVerifier* (CV) is defined, that takes a set of routes $\mathcal{R}$, the start times in Γ, and the current set of paths CP as input and returns:

- *CFS*, a list that expresses where each vehicle is at each time; this is empty if the problem is infeasible.
- $\bar{\mathcal{C}}$, the *Unsat Core* relative to constraints (E.5)-(E.7) (see Section 3.3); this is empty if the problem is feasible.

3.2 Paths Changing Problem

In the *Paths Changing Problem*, alternative paths are computed to connect the consecutive customers of each route. Finding alternative paths may be necessary when, for a given set of routes $\mathcal{R}$ and starting times Γ, no feasible schedule exists. The *Capacity Verification Problem* may be infeasible due to the current set of paths that connect the customers' locations, therefore a different set may lead to a feasible solution. A route is defined as a sequence of customers, and for any two consecutive customers there is a path (a sequence of edges) connecting them. Therefore, for a route containing $i+1$ customers we will have i paths and for each path we can define a start and an end node, ξ_i and π_i, respectively. The sets of outgoing and incoming edges for a certain node n are denoted $\mathcal{O}_n$ and $\mathcal{I}_n$, respectively.

Decision variables used to build the model are:

w_{rpn}: Boolean variable that represents whether the pair p of route r is using node n;

z_{rpe}: Boolean variable that represents whether the pair p of route r is using edge e;

This problem can be split into $r \cdot i$ sub-problems (assuming all routes have $i+1$ customers) that find paths for each route separately; simpler and smaller models are faster to solve. Unfortunately it may be necessary to explore different combinations of paths, so to retain the information we have only one model. Therefore, let the optimal solution to the *Path Changing Problem* found at iteration h be

$$CP = \bigcup_{\substack{r \in \mathcal{R} \\ p \in \mathcal{P}_r \\ e \in \mathcal{E}}} \{z^*_{rpe}\},$$

where z^*_{rpe} is the value of z_{rpe} in the current solution; also, let PP be the set containing the optimal solutions found until the $(h-1)$-th iteration.

The model is then:

$$\min_{r\in\mathcal{R},\ p\in\mathcal{P}_r,\ n\in\mathcal{E}} \sum \mathrm{If}(z_{rpe}, d_e, 0) \tag{E.8}$$

$$w_{rp\xi_p} \wedge w_{ri\pi_p}, \qquad \forall p \in \mathcal{P}_r,\ r \in \mathcal{R} \tag{E.9}$$

$$\mathrm{EN}_{e\in\mathcal{O}_{\xi_p}}(z_{rpe}, 1), \qquad \forall p \in \mathcal{P}_r,\ r \in \mathcal{R} \tag{E.10}$$

$$\mathrm{EN}_{e\in\mathcal{I}_{\xi_p}}(z_{rpe}, 1), \qquad \forall p \in \mathcal{P}_r,\ r \in \mathcal{R} \tag{E.11}$$

$$z_{rpe} \implies \neg z_{rp\bar{e}}, \qquad \forall p \in \mathcal{P}_r,\ r \in \mathcal{R},\ e \in \mathcal{E} \tag{E.12}$$

$$\bigwedge_{n\in\mathcal{N}, n\neq\xi_p, n\neq\pi_p} \mathrm{If}(w_{rpn}, \mathrm{EN}_{e\in\mathcal{O}_n}(z_{rpe}, 1) \wedge \mathrm{EN}_{e\in\mathcal{I}_n}(z_{rpe}, 1), \mathrm{EN}_{e\in\mathcal{O}_n}(z_{rpe}, 0) \wedge \mathrm{EN}_{e\in\mathcal{I}_n}(z_{rpe}, 0)), \quad \forall p \in \mathcal{P}_r,\ r \in \mathcal{R} \tag{E.13}$$

$$\bigvee_{z_{rpe}\in CP} \neg z_{rpe}, \qquad \forall CP \in PP \tag{E.14}$$

The cost function (E.8) to minimize is the cumulative length of the used edges; (E.9) guarantees that, for each path of each route, the start and end nodes are used; (E.10) and (E.11) make sure that exactly one outgoing (incoming) edge is incident with the start (end) node of a route; (E.12) makes sure that a path is not allowed to use both an edge and its reverse; (E.13) guarantees that if a node (different from the start or end) is selected, exactly one of its outgoing and one of its incoming edges will be used. On the other hand, if a node is not used, none of its incident edges will be used; finally, (E.14) rules out all the previously found solutions.

Based on the model described above the function *PathsChanger* (PC) is defined, that takes the previous paths PP as input and returns a new set of paths NP. If the *Paths Changing Problem* is infeasible then $NP = \varnothing$.

Up to this point, unless specified otherwise, the models presented are taken from [11].

3.3 Exploiting the *MUC*

Experiments reported in [11], show that ComSat performs well for many problem instances, however, for some specific instances ComSat failed to find feasible solutions in reasonable time. Investigations revealed the *PC* to be the culprit. The reason is that it searches blindly through the possible paths that connect any two customers, while minimizing the paths' cumulative length. A *conflict-free* solution may involve paths that are quite longer than the current ones though, and the *PC* will have to explore many *shorter* solutions before finding the right one. Improving the performance of the *PathsChanger* would be beneficial for the overall performance of ComSat, and letting the *MUC* guide the paths changing is such an improvement.

When extracting the *MUC*, it is possible to only track specific constraints. This feature can be exploited to focus only on the capacity constraints violations. In fact, since time windows and service time are not flexible, it is of little to no use to track constraints represented by (E.1)-(E.4). Also, an infeasible formula φ may have multiple *MUCs*; in the CF-EVRP this means that conflicts may arise at different locations in the plant. In order to catch all of them, it is possible to iteratively relax the conflicting constraints from the initial formula and solve it again, until it becomes feasible. The formula will indeed become feasible eventually, since it is based on a feasible solution $\mathcal{R}$ and only the capacity constraints can make it infeasible; in the worst case all such constraints will be removed during the iterations. Note that, since not all constraints are tracked, the set of constraints $\bar{\mathcal{C}}$ returned is not an actual *Unsat Core*, since $\bar{\mathcal{C}}$ would only make the problem infeasible in conjunction with the untracked constraints. Nonetheless, it provides the information about the conflicts needed to guide the search of paths.

Let φ_0 be the conjunction of constraints (E.1)-(E.7). Assume that φ_0 is infeasible, and let $\bar{\mathcal{C}}_0$ be the subset of a *MUC* retrieved by tracking constraints (E.5)-(E.7). Then let $\varphi_1 = \varphi_0 \setminus \bar{\mathcal{C}}_0$, also infeasible, and let $\bar{\mathcal{C}}_1$ be the subset of a *MUC* retrieved by tracking constraints defined by (E.5)-(E.7), not including the ones in $\bar{\mathcal{C}}_0$. In general, the constraints in $\bar{\mathcal{C}}_{i-1}$ can be iteratively relaxed to obtain a new formula φ_i, until a feasible $\varphi_n = \varphi_0 \setminus (\bar{\mathcal{C}}_0 \cup \ldots \cup \bar{\mathcal{C}}_{n-1})$ is found. Then $\bar{\mathcal{C}} = \bar{\mathcal{C}}_0 \cup \ldots \cup \bar{\mathcal{C}}_{n-1}$ contains all the conflicts due to the capacity constraints.

Each constraint represented by (E.5)-(E.7) is defined over two routes r_1 and r_2 and their pairs p_1 and p_2 for a specific node n or edge e; therefore, if the

constraint is part of $\bar{\mathcal{C}}$, the routes and pairs that caused the conflict over n or e can be identified. If the conflict was generated by a constraint from (E.5), then the following constraint is added to (E.8)-(E.14):

$$\neg(w_{r_1 p_1 n}) \vee \neg(w_{r_2 p_2 n}). \tag{E.15}$$

On the other hand, if the conflict was caused by constraint from (E.6) or (E.7), the following constraint is added to (E.8)-(E.14):

$$\neg(z_{r_1 p_1 e}) \vee \neg(z_{r_2 p_2 e}). \tag{E.16}$$

Constraints (E.15) and (E.16) force at least one of the routes involved in the conflict to avoid the specific node (edge, respectively) when computing a path for the pairs involved in the conflict. The constraint is formulated so that the choice of the route to change is left to the solver, including the possibility of changing both routes; since the problem is an optimization, the solver will choose the change that leads to the shortest cumulative paths length.

Based on the model described by (E.8)-(E.16), the function *MUC-Guided-Paths-Changer* (*GPC*) is defined, that takes the previous paths *PP* and $\bar{\mathcal{C}}$ as input and returns a new set of paths *NP*. If the *Path Changing Problem* is infeasible $NP = \varnothing$.

Since for each constraint in $\bar{\mathcal{C}}$ a new constraint is added to the *GPC*, it is imperative that the *Unsat Core* returned when the *CV* is infeasible is *minimal*. This is so because if the *Unsat Core* is not minimal, it could contain constraints that are not actually causing *capacity conflicts*. These constraints would in turn lead to defining constraints (E.15) and (E.16) in the *GPC* that may remove feasible solutions.

Fig. 2 summarizes the steps required to find a conflict-free schedule *CFS*, if such exists, using the improved paths searching algorithm *GPC*. As mentioned, it is assumed that routes $\mathcal{R}$ and their start times Γ have already been computed. The shortest paths between any two customers are computed using Dijkstra's algorithm and then set as the current paths *CP* to travel among customers. Also, *CP* are added to the list of previous paths *PP*.

Then the *CV* will check such routes against the capacity constraints; if this sub-problem has a feasible solution the algorithm terminates and a conflict-free schedule is returned. Otherwise $\bar{\mathcal{C}}$ is extracted as described in the previous paragraph and the the *GPC* algorithm is invoked. *GPC* will use the informa-

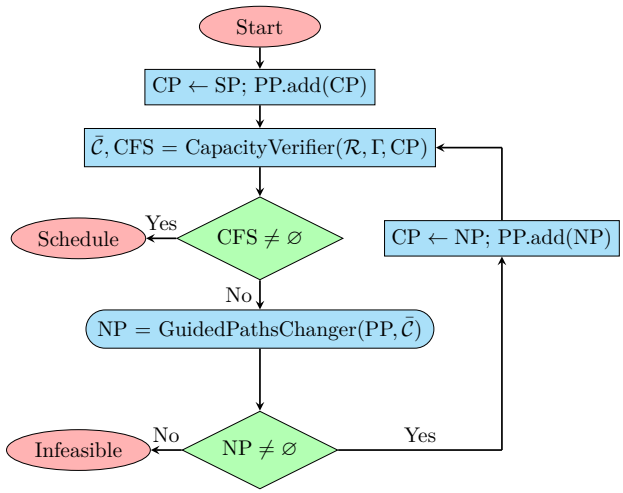

Figure 2: Flowchart of the *MUC*-Guided-CFPS.

tion about previously computed paths *PP* and the information about conflicts from $\bar{\mathcal{C}}$ to compute new paths *NP*, which will be set as the current paths and stored in *PP*. At this point the *CV* is run again using the new paths. The iterations between the two algorithms continue until either the *CV* is feasible, or the *GPC* is infeasible, i.e., there are no feasible, conflict-free paths to execute the routes $\mathcal{R}$ with the start times Γ.

4 Proof of Soundness and Completeness

In this section, proof of soundness and completeness of the *Unsat Core* Guided CFPS is provided. The underlying idea for the proof is the following. There exists a finite number of solutions to the *Path Changing Problem*; the *GPC* can enumerate at least all feasible solutions to the *Path Changing Problem*; if a solution that satisfies the *Capacity Constraints* does exists, the *GPC* will eventually find it, otherwise it will declare the problem infeasible.

Let $\mathcal{S}$ be the set of possible solutions to a *Path Changing Problem*; let us divide $\mathcal{S}$ into the set of conflict-free solutions $\mathcal{F}$ and the set of conflicting solutions $\mathcal{U}$. In other words a solution to the *Path Changing Problem* from $\mathcal{F}$ will make the *Capacity Verification Problem* feasible, while a solution from $\mathcal{U}$

will not. If the CFPS is infeasible, then $\mathcal{S} = \mathcal{U}$ and $\mathcal{F} = \varnothing$. In this case, even if the *GPC* is not able to find all feasible solutions $\mathcal{F}$, there is none to find.

In case the CFPS is feasible though, in order to prove completeness it is necessary to guarantee that at least all feasible solutions $\mathcal{F}$ can be found by *GPC*. This is proven for the *PC*, since each call of the *PC* function will find the next optimal solution to the *Path Changing Problem*, whether it belongs to $\mathcal{F}$ or not, until all solutions are enumerated. However in the *GPC* there are additional constraints that may remove feasible solutions. In the proof it is shown that such additional constraints only remove infeasible solutions.

Observation 3. *The* Path Changing Problem *is a satisfiability problem in propositional logic. The* Capacity Verification Problem *falls into the category of difference logic (a fragment of linear arithmetic). Thus, both problems are decidable.*

Observation 4. *The* Path Changing Problem *is bounded. In fact, the* Path Changing Problem *involves only a finite number of Boolean variables, so its domain is finite.*

Lemma 1: *Given a finite, directed, weighted graph, the number of paths that connect two arbitrary nodes is finite.*

Proof. By definition, a path is an ordered set of nodes such that no node appears more than once. If the number of nodes in the graph is finite, there cannot be an infinite number of paths. □

Lemma 2: *For a given set of routes $\mathcal{R}$ and start times in Γ, repeated calls to the PC function will enumerate all feasible solutions to the* Path Changing Problem*, either belonging to $\mathcal{F}$ or $\mathcal{U}$, before returning infeasible.*

Proof. Let φ_0 be the conjunction of constraints (E.9)-(E.13), a relaxation of the *Paths Changing Problem*, and let CP_0 be a solution to φ_0. Then, if another solution CP_1 for φ_0 exists, it can be found by solving $\varphi_0 \wedge \neg CP_0 = \varphi_1$. In general, the n-th solution can be found by solving $\varphi_0 \wedge \neg CP_0 \wedge \ldots \wedge \neg CP_{n-1} = \varphi_n$. Because of *Lemma* 1, we know that the number of solutions to the *Paths Changing Problem*, $|\mathcal{S}|$, is finite and we can enumerate them all by solving $\varphi_0, \ldots, \varphi_{|\mathcal{S}|-1}$. □

Lemma 3: *Using the PC and* CV *is a sound and complete procedure to solve the CFPS*

Proof. Because of Observation 1 we know there is a finite number of solutions to the *Path Changing Problem*, and because of Lemma 2 we know that the *PC* function can enumerate them all. If a solution that belongs to $\mathcal{F}$ exists the *PC* will find it, otherwise it will return all solutions belonging to $\mathcal{U}$; the *CV* will then check whether they are conflict-free. Therefore, using the *PC* and *CV* in combination will correctly solve the CFPS. □

Lemma 4: *For a given set of routes $\mathcal{R}$, the GPC is able to find at least all solutions in $\mathcal{F}$.*

Proof. For each set of current paths *CP*, $\bar{\mathcal{C}}$ only contains constraints defined by (E.5), (E.6), and (E.7). The constraints in $\bar{\mathcal{C}}$ are iteratively retrieved from *minimal Unsat Core* and therefore represent combinations of nodes and edges in the graph where the conflicts happen. Since each constraint defined by (E.15) and (E.16) addresses one constraint from $\bar{\mathcal{C}}$, (E.15) and (E.16) only define constraints over nodes or edges that cause conflicts. Hence these constraints only remove solutions of the *Path Changing Problem* that belong to $\mathcal{U}$. □

Theorem 1: *Using the GPC and* CV *is a sound and complete procedure to solve the CFPS.*

Proof. The *PC* and the *GPC* are identical, except for constraints (E.15)-(E.16), and because of Lemma 4, we know that the addition of these constraints only removes solutions from $\mathcal{U}$. Thus, since the CFPS using the *PC* is sound and complete (Lemma 3), so is the CFPS using the *GPC*. □

5 Experiments

In order to evaluate the goodness of the proposed method and its performance against the previous version of the CFPS algorithm, a set of problem instances is designed and used for testing. Both the *PC* and *GPC* are embedded in the ComSat algorithm. However, since the goal is to compare the search for alternative paths, problems are designed in such a way that there is only one feasible set of routes $\mathcal{R}$ to serve the customers; also, only the running time for search of conflict-free paths is measured. The algorithms called by ComSat

used the SMT solver Z3 4.8.9 to solve the models. All the experiments[1] were performed on an *Intel Core i7 6700K, 4.0 GHZ, 32GB RAM* running *Ubuntu-18.04 LTS*.

Table 1 shows the results of the evaluation of five problem instances of the CF-EVRP solved using ComSat. Each instance was solved twice, once using the *PC* and once using the *GPC*; in each case the number of iterations and the time (in seconds) required to find a feasible solution is reported. The problem instances presented are increasingly hard to solve, in terms of plant size (represented by the number of nodes), number of routes and number of customers in each route. The customers' locations and time windows so that conflicts will arise due to the capacity constraint when the shortest paths are used and a search for alternative paths will be necessary in order to find a conflict-free schedule.

For instances *1* through *4* it took only one iteration to the *GPC* to find a feasible solution, while the *PC* required an increasing number of iterations to find a feasible solution, as the instances grew more complicated. The gap in the running time between the *GPC* and the *PC* follows the same trend; for instance *1* it only takes 2 iterations to the *PC* to find a feasible solution, while it takes 24 and 54 iterations to find a solution to instances *2* and *3*. This number drops to 15 iterations for instance *4*. On average, a single iteration of the *PC* takes less time than an iteration of the *GPC*, but due to the larger number of iterations required, the overall running time for the *PC* is always larger.

Instance *5* is the odd one out, as it only takes one iteration of the *PC* to find a feasible solution, and, as for the other instances, the running time for the single iteration is shorter.

Results and Discussion

The experiments show that for most of the instances the *GPC* performed better than *PC* in terms of running time and number of iterations. To be more specific, one iteration of the *GPC* is slower than one iteration of the *PC*, but the number of iterations required by the *PC* is always higher, and therefore the overall execution time is longer. As the instances become larger, the gap between the running time for one iteration of each method increases too.

[1]The implementation of the *GPC* presented in Section 3.3 and the problem instances are available in the *UNSAT_Core* folder at `https://github.com/sabinoroselli/VRP.git`.

However, since the number of iterations required for more complex instances grows as well, the *GPC* shows increasing good performance for harder-to-solve instances. On the other hand, Instance *5* shows a different result, since both the *PC* and the *GPC* take only one iteration. As for the other instances, a single iteration of the *PC* is faster, hence the *PC* beats the *GPC* on Instance *5*. We can conclude that for some instances, the *PC* may be able to quickly find feasible solutions and outperform the *GPC*. However this is behaviour is highly dependent on the instance and as instances grow larger the chances could grow smaller, as the number of possible paths available increases. Moreover, a detailed analysis of the solutions to the *Path Changing Problem* for each instance[2] confirms that, for the *PC*, there is no convergence to a feasible solution as the number of iterations increases, since the number of conflicts does not always decrease at the following iteration. On the other hand, the *GPC* shows a consistent behaviour as it always takes only one iteration to find feasible solutions.

Table 1: Comparison of the *PC* and *GPC* over a set of instances of the CF-EVRP. For each instance the number of iterations and the total running time (in seconds) required to find a feasible solution is reported.

Inst.	$\|\mathcal{N}\|$	$\|\mathcal{R}\|$	$\|\mathcal{K}\|$	Iterations		Time	
				PC	*GPC*	*PC*	*GPC*
1	3	2	4	2	1	0.25	**0.16**
2	8	3	6	24	1	8.81	**0.40**
3	5	4	8	54	1	35.92	**1.08**
4	64	4	28	15	1	643.40	**184.60**
5	64	4	28	1	1	**21.20**	128.40

6 Conclusions

This paper presents an algorithm to search for conflict-free paths for a set of routes to serve customers in a conflict-free electric vehicle routing problem (CF-EVRP). The algorithm exploits the SMT solvers' ability to return a

[2]Details of the problem instances are discussed in the file *Instances_Results.pdf* in the *UNSAT_Core* folder of the Github repository.

MUC when a formula is infeasible, to guide the search for paths. Soundness and completeness of the algorithm are proved, and preliminary experimental data based on a set of generated CF-EVRP problem instances are provided. The experiments show that the new *MUC* based algorithm consistently finds feasible paths taking only one iteration and significantly shorter time than the previous naive method. Future work includes to run extensive computational analyses to strengthen the claims made in this paper, and further development of the *MUC* guided paths search by improving the information extraction from the *MUC*.

References

[1] N. N. Krishnamurthy, R. Batta, and M. H. Karwan, "Developing conflict-free routes for automated guided vehicles," *Operations Research*, vol. 41, no. 6, pp. 1077–1090, 1993.

[2] A. I. Corréa, A. Langevin, and L.-M. Rousseau, "Scheduling and routing of automated guided vehicles: A hybrid approach," *Computers & operations research*, vol. 34, no. 6, pp. 1688–1707, 2007.

[3] M. Saidi-Mehrabad, S. Dehnavi-Arani, F. Evazabadian, and V. Mahmoodian, "An ant colony algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs," *Computers & Industrial Engineering*, vol. 86, pp. 2–13, 2015.

[4] R. Yuan, T. Dong, and J. Li, "Research on the collision-free path planning of multi-AGVs system based on improved A* algorithm," *American Journal of Operations Research*, vol. 6, no. 6, pp. 442–449, 2016.

[5] E. Thanos, T. Wauters, and G. Vanden Berghe, "Dispatch and conflict-free routing of capacitated vehicles with storage stack allocation," *Journal of the Operational Research Society*, pp. 1–14, 2019.

[6] K. Murakami, "Time-space network model and MILP formulation of the conflict-free routing problem of a capacitated AGV system," *Computers & Industrial Engineering*, 2020.

[7] M. Zhong, Y. Yang, Y. Dessouky, and O. Postolache, "Multi-AGV scheduling for conflict-free path planning in automated container terminals," *Computers & Industrial Engineering*, vol. 142, 2020.

[8] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor, and P. J. Stuckey, "Integrated task assignment and path planning for capacitated multi-agent pickup and delivery," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5816–5823, 2021.

[9] S. F. Roselli, M. Fabian, and K. Åkesson, "Solving the conflict-free electric vehicle routing problem using SMT solvers," in *2021 29th Mediterranean Conference on Control and Automation (MED)*, IEEE, 2021, pp. 542–547.

[10] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[11] S. Roselli, M. Fabian, and K. Åkesson, "A compositional Algorithm to solve the Conflict-Free Electric Vehicle Routing Problem," *2022 IEEE Transactions on Automation Science and Engineering. Submitted for Publication*, pp. 1405–1421, 2022.

[12] C. Barrett and C. Tinelli, "Satisfiability modulo theories," *Handbook of model checking*, pp. 305–343, 2018.

[13] L. De Moura and N. Bjørner, "Satisfiability modulo theories: Introduction and applications," *Commun. ACM*, vol. 54, no. 9, pp. 69–77, Sep. 2011, ISSN: 0001-0782.

[14] T. Weber, S. Conchon, D. Déharbe, M. Heizmann, A. Niemetz, and G. Reger, "The SMT competition 2015–2018," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 11, no. 1, pp. 221–259, 2019.

[15] R. Sebastiani and P. Trentin, "OptiMathSAT: A tool for optimization modulo theories," *Journal of Automated Reasoning*, vol. 64, no. 3, pp. 423–460, 2020.

[16] J. L. Gross and J. Yellen, *Handbook of graph theory*. CRC press, 2003.

[17] A. Cimatti, A. Griggio, and R. Sebastiani, "Computing small unsatisfiable cores in satisfiability modulo theories," *Journal of Artificial Intelligence Research*, vol. 40, pp. 701–728, 2011.

[18] D. Selsam and N. Bjørner, "Guiding high-performance SAT solvers with unsat-core predictions," in *International Conference on Theory and Applications of Satisfiability Testing*, Springer, 2019, pp. 336–353.

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[20] N. Dershowitz, Z. Hanna, and A. Nadel, "A scalable algorithm for minimal unsatisfiable core extraction," in *International Conference on Theory and Applications of Satisfiability Testing*, Springer, 2006, pp. 36–41.

[21] J. Huang, "MUP: A minimal unsatisfiability prover," in *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, 2005.*, IEEE, vol. 1, 2005, pp. 432–437.

[22] A. Nadel, "Boosting minimal unsatisfiable core extraction," in *Formal Methods in Computer Aided Design*, IEEE, 2010, pp. 221–229.

[23] D. Kroening and O. Strichman, *Decision procedures - An Algorithmic Point of View*. Springer, 2016.

[24] A. Nadel, V. Ryvchin, and O. Strichman, "Efficient MUS extraction with resolution," in *2013 Formal Methods in Computer-Aided Design*, IEEE, 2013, pp. 197–200.

[25] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νZ-an optimizing SMT solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2015, pp. 194–199.

[26] S. F. Roselli, K. Bengtsson, and K. Åkesson, "SMT solvers for job-shop scheduling problems: Models comparison and performance evaluation," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2018, pp. 547–552.

[27] F. A. Aloul, B. Al Rawi, and M. Aboelaze, "Identifying the shortest path in large networks using boolean satisfiability," in *2006 3rd International Conference on Electrical and Electronics Engineering*, IEEE, 2006, pp. 1–4.

[28] C. Sinz, "Towards an optimal CNF encoding of boolean cardinality constraints," in *International conference on principles and practice of constraint programming*, Springer, 2005, pp. 827–831.