KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Formal, exact and metaheuristic methods for combinatorial optimization

Tu-San Pham

Supervisors:
Prof. dr. Patrick De Causmaecker
Prof. dr. Marc Denecker

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

February 2020

Formal, exact and metaheuristic methods for combinatorial optimization

Tu-San PHAM

Examination committee:
Prof. dr. ir. Omer Van der Biest, chair
Prof. dr. Patrick De Causmaecker, supervisor
Prof. dr. Marc Denecker, supervisor
Prof. dr. ir. Greet Vanden Berghe
Prof. dr. ir. Gerda Janssens
Prof. dr. Frits Spieksma
Prof. dr. Louis-Martin Rousseau
(Polytechnique Montréal, Canada)
Prof. dr. Laetitia Jourdan
(Université Lille 1, France)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

February 2020

© 2020 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Tu-San Pham, Etienne Sabbelaan 53 box 7659, 8500 Kortrijk (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

There are more things in Heaven and Earth, Horatio,
Than are dreamt of in your philosophy.

— William Shakespeare, *Hamlet*

Preface

If this dissertation is the summary of my professional journey in the last five years of my PhD, this small section is devoted to something not any less important - my personal journey during the last six years in Belgium, including also the first year as a pre-doc student. Those six incredible years have seen me growing up and transformed me into the person I am today, and I am thankful to every person that I met along the way.

First and foremost, I would like to express my gratitude to my supervisor, Patrick. Thank you for guiding me, encouraging me and most importantly, believing in me even when I doubted myself. My favourite parts of our meetings were when you ask "are you doing okay" and when you talked eagerly about physics and sometimes even Shakespeare even though I didn't understand most of them. I am truly grateful for having you as my supervisor. I would like to thank my co-supervisor, Marc. You have helped me more than you think and I really appreciate it. I would also like to thank thầy Tiến and Vivian Gascon, who played important roles in the early parts of my career.

To Jo Devriendt, it is difficult to express how grateful I am to you. You were not only a wonderful co-worker but also a mentor to me, especially during my first years in Belgium. Thank you for being so patient with my numerous questions and also my bad English. Your pronunciation lessons are still among the best lessons that I have ever had and "comptine d'un autre été" is still the only real song that I can play up till now.

Thank you Louis-Martin for your advice during my wonderful time in Montréal, for flying all the way to Belgium for my defense, and for giving me a chance to continue my professional career under your guidance. Thank you Greet for your valuable comments which have helped me improve the quality of my dissertation. I am grateful for Gerda Janssens, Frits Spieksma, and Laetitia Jourdan for finding time in their busy schedule to read my dissertation, to provide feedback and suggestions, and to come to my defense. I also thank Omer Van der Biest for agreeing to chair my private and public defenses.

I would like to thank all my colleagues. Thank you chị Nguyên and anh Hoàng, my two Vietnamese colleagues and also friends who have helped me a lot not only in my

professional life but also in my personal life. Thanks Hans, Ayman, Morteza, Tommy, Pieter, Chao, Jorik, Chen and Jens for sharing nice discussions and nice moments at work. I would like to thank my fellow students in the STEEP project who have helped me a lot in the first two tough years of my PhD.

It can be hard and lonely being a foreign student living in a foreign country. I am lucky enough to have friends who I can see as my second family, who have been with me and supported me through the toughest moments. Caroline, Iwanna, My, Trâm, Quỳnh, Linh, Gertrude, Qinxiao, Antoine ... it is difficult to name you all but I would like to let you know how thankful I feel for having you as friends. Especially, thank you Quân, Toàn, and Mohcine. I forever appreciate your support.

Along the way, I had the chance to meet and become friends with many wonderful people. I would like to thank all my friends in Kulak, Vietnam, Kortrijk, Leuven, Ghent, Brussels, Montréal and more. Thanks for all the laughter, all the trips, all the glimmering memories that we shared, that I shall never forget. You add joyful colours to my rainbow.

My deepest appreciation goes to my family. Bố, mẹ, chị Thu, thank you for your unconditional love, for your continuous support during my whole life, and for letting me know that I always have a home to come back to.

Finally, and non-conventionally, I would like to express my gratitude to everything that I have, everyone that I met in life. Despite of all the ups and downs, I consider myself as one of the luckiest persons on earth :).

Brussels, February 2020

Phạm Tú San

Abstract

Combinatorial optimization problems are ubiquitous in real life and hence a wide range of solving paradigms are available. Each paradigm has its own characteristics, which are potentially complementary. Therefore, investigating the possibility to combine different approaches might be beneficial for solving combinatorial problems. This dissertation exploits this research theme at two different levels.

The first part of this dissertation exploits the relationships between solving paradigms at a general level. We investigate the combination of two sub-domains of Operations Research and Artificial Intelligence, namely Local Search and Knowledge Representation. We propose "declarative local search", which allows for specifying local search heuristics declaratively. Declarative local search is built on top of IDP, a Knowledge Base System. IDP consists of a set of inference methods for solving different tasks around a center knowledge base described in the formal language FO($\cdot$), an extension of first-order logic. Declarative local search enables local search heuristics to be synthesized from their formal descriptions. The framework has been proven empirically to be able to serve as a fast prototype tool for local search heuristics and also to function as an alternative back-end for some combinatorial problems that are traditionally difficult for formal systems such as IDP.

In the second part, we study two particular combinatorial problems and how different solving methods are utilized to solve them. The first case study, namely the Intermittent Travelling Salesman Problem, presented in chapter 4, is a new variant of the Travelling Salesman Problem. The problem is inspired by real-world drilling/texturing applications where the temperature of the work-piece is taken into account. An exact branch-and-bound approach and four Variable Neighbourhood Search metaheuristics are proposed. The problem's characteristics are analyzed and an instance library is created and made publicly available for future research. The second case study, namely the Radiotherapy Scheduling Problem, presented in chapter 5, is a real scheduling problem at CHUM, a large cancer center in Montréal, Canada. We propose a two-phase approach where Mixed Integer Programming and Constraint Programming models are proposed for each phase. The algorithm is tested on a realistic dataset generated from real data

provided by CHUM. The results show the dominance of a non-conventional Constraint Programming approach over the conventional Mathematical Programming for the problem.

Summarized, the two parts of this dissertation investigate the relationships between solving paradigms at two complementary levels. We investigate the combined use of and the interaction between very different methodologies in a common domain of application.

Beknopte samenvatting

Problemen van combinatorisch optimalisatie komen overvloedig voor in het dagelijkse leven. Mede hierdoor is een breed spectrum aan oplossingsmethodes en -paradigma's voorhanden. Elk paradigma heeft zijn eigen karakteristieken, die in veel gevallen complementair zijn. Daarom is het interessant om de mogelijkheden te onderzoeken om methodes van verschillende paradigma's te combineren. Deze verhandeling gaat op dit onderzoeksthema dieper in en ontwikkelt het op twee niveaus.

Het eerste deel van de verhandeling behandelt de verbanden tussen oplossingsparadigma's op een algemeen niveau. We onderzoeken de combinatie van twee sub domeinen van Operationeel Onderzoek en Artificiële intelligentie, namelijk Local Search en Kennisrepresentatie. We stellen een "Declaratieve Local Search" voor die toelaat om local search heuristieken declaratief te specifiëren. Declaratieve Local Search bouwt op IDP. IDP is een Kennisbank systeem met een referentie engine di verschillende taken kan oplossen rond een centrale kennisbank die in de formele taal FO($\cdot$) geschreven is. FO($\cdot$) is een uitbreiding van eerste orde logica. Het laat toe om local search heuristieken te synthetiseren uit de formele beschrijvingen. Dit systeem wordt empirisch gevalideerd en er wordt experimenteel gedemonstreerd dat toelaat snel prototype voor local search heuristieken te genereren. Het kan ook werken als een alternatieve backend voor sommige combinatorische problemen die anders moeilijk zijn voor formele systemen zoals IDP.

In het tweede deel bestuderen we twee specifieke problemen via verschillende oplossingsmethodes. De eerste gevallenstudie gaat over het "Intermittent Travelling Salesman Problem" (ITSP). Zoals beschreven in hoofdstuk 4 is dit een nieuwe variant van het "Travelling Salesman Problem". Het is geïnspireerd op problemen die optreden bij het toepassingen in het bewerken van oppervlakten door te boren of puntsgewijs te bewerken met bijvoorbeeld lasers. De temperatuur van het werkstuk is hierbij in rekening te nemen. Een exacte branch-and-bound methode en vier verschillende Variable Neighbourhood Search metaheuristieken worden voorgesteld. De karakteristieken van het probleem worden geanalyseerd en een bibliotheek van instanties wordt gecreëerd. Deze laatste is publiek beschikbaar voor verder

onderzoek. De tweede gevallenstudie betreft het Radiotherapy Scheduling Problem. Zoals beschreven in hoofdstuk 5 is dit probleem gebaseerd op een werkelijke situatie in CHUM, een groot centrum voor kankerbehandeling in Montréal, Canada. We stellen een tweefasen benadering voor waarbij Mixed Integer Programming en Constraint Programming worden gecombineerd. Het algoritme is getest op een werkelijke dataset de gegenereerd werd uit de gegevens bij CHUM. Het resultaat toont dat een niet-conventionele Constraint Programming benadering de beste is.

Samengevat onderzoeken de twee delen van deze verhandeling het verband tussen oplossingsparadigma's op twee complementaire niveaus. We bestuderen het gezamenlijk gebruik van en de interactie tussen zeer verschillende methodologien in eenzelfde toepassingsdomein.

List of Abbreviations

AI Artificial Intelligence.

ASP Answer Set Programming.

BNB Branch-and-Bound.

CBLS Constraint-Based Local Search.

CP Constraint Programming.

GVNS General Variable Neighborhood Search.

IDP Imperative-Declarative Programming.

IP Interger Programming.

ITSP Intermittent Travelling Salesman Problem.

KBS Knowledge Base System.

KR Knowledge Representation.

LNS Large Neighbourhood Search.

LP Logic Programming.

MIP Mixed Interger Programming.

MP Mathematical Programming.

OR Operations Research.

RTSP Radiotherapy Scheduling Problem.

SAT Satisfiability checking of propositional logic.

TSP Travelling Salesman Problem.

VND Variable Neighborhood Descent.

VNS Variable Neighborhood Search.

Contents

Abstract v

Beknopte samenvatting vii

Contents xi

List of Figures xv

List of Tables xvii

1 Introduction 1

1.1 Categorizing of solving paradigms for combinatorial optimization problems 2

1.1.1 Operations Research and Artificial Intelligence 2

1.1.2 Exact and heuristic methods 3

1.2 State-of-the-art and research questions 3

1.3 Structure of the dissertation 7

I Declarative local search in predicate logic 11

2 Knowledge Representation and Local Search-based heuristics 12

2.1 Knowledge Representation and Knowledge Base System 12
2.1.1 Knowledge Representation 12
2.1.2 Knowledge Base System 14
2.2 Combinatorial optimization problems and local search-based heuristics 15
2.3 Motivation: the bridge between Declarative Knowledge Representation and Local Search heuristics 16
2.4 Declarative local search 18
2.5 Related work 19
2.5.1 Local search in Logic Programming 19
2.5.2 Local search in Constraint Programming 21
2.5.3 Other paradigms 23
2.5.4 Discussion 23

3 Declarative Local Search in IDP 25

3.1 Introduction 26
3.2 IDP and FO(·) 27
3.3 Modelling local search heuristics 29
3.3.1 Modelling neighbourhood moves 29
Standard neighbourhood moves 29
LNS moves 32
3.3.2 Local search algorithms in IDP 33
3.4 Local search framework 34
3.5 Experimental results 37
3.6 Discussion 39
3.7 Conclusion and future work 40

II Hybrid algorithms for specific case studies **43**

4 The Intermittent Travelling Salesman Problem **44**

4.1 Introduction 44
4.2 Problem description 46
4.3 Branch-and-bound approach 48
4.3.1 Mathematical models and lemmas 48
Subproblem 1: a fixed number of processing-steps 48
Subproblem 2: a fixed sequence of visits 49
4.3.2 Branch-and-bound approach 52
4.4 Variable Neighborhood Search 53
4.4.1 Solution encoding 55
4.4.2 Initial solution constructor 56
4.4.3 Local search 56
4.4.4 Perturbation 59
4.4.5 The GVNS for the ITSP 61
4.5 Experiment 62
4.5.1 Test instances 62
4.5.2 Evaluation of the branch-and-bound approach 63
4.5.3 Calibration of the metaheuristics 68
4.5.4 Results and metaheuristics comparison 68
4.6 Conclusion and future work 71

5 The Radiotherapy Scheduling Problem **73**

5.1 Introduction 74
5.2 Problem description 77
5.3 A two-phase approach 82

5.3.1 Phase 1: assigning patients to days and linacs 82
5.3.2 Phase 2: ordering patients on linacs - IP model 83
5.3.3 Phase 2: order of patients on linacs - CP model 85
5.4 Experiment 87
5.4.1 Data description 87
5.4.2 Data generation 87
5.4.3 Numerical results 88
5.5 Conclusions and future work 90

6 Conclusion **91**
6.1 Summary and contributions 91
6.2 Future research directions 93

Bibliography **95**

List of publications **109**

List of Figures

3.1 The 2-opt move. 32

3.2 The built-in local search algorithms in IDP. 34

3.3 The local search framework in IDP. 36

3.4 Creating a neighbouring solution in IDP. 36

3.5 Results for the TSP. 39

4.1 Sample instance illustrating a larger number of processing-steps may yield a better solution. 47

4.2 A fixed sequence of visits. 50

4.3 Solution encoding for an instance with 3 vertices. The solution includes 5 processing-steps where vertex A is visited three times. 56

4.4 Move evaluation example. 59

4.5 Hyperedges exchange. 60

4.6 Branch-and-bound performance on different subsets. 66

4.7 Results on special instances of set *T1*. 69

5.1 Percentages of patients receiving treatments within the recommended deadline by states in Canada (2018).[1] 75

5.2 Percentages of patients receiving treatments before the recommended deadline in Quebec. [2] 75

List of Tables

2.1 CBLS systems. 22

4.1 Density setting configuration. 63
4.2 Instance configuration of set R. 63
4.3 Branch-and-bound results on the three subsets *R1*, *R2* and *R3* respectively. 65
4.4 Detailed results on some selected instances in set *R*. 67
4.5 Parameters tuning. 68
4.6 The VNS performance on set *R*. 69
4.7 Metaheuristics performance on set *T2*. 70

5.1 Distribution of patient categories. 87
5.2 Results with 30 seconds time limit. 89
5.3 Results with 1200 seconds time limit. 89

Chapter 1

Introduction

In this chapter, we situate the problem, describe the current state of the art and define the main objectives of the dissertation.

Combinatorial optimization, formally, consists of finding an optimal object from a finite set of objects. One of the simplest examples of combinatorial optimization problems is the shortest path problem, which consists of finding the shortest path between two nodes in a graph. Combinatorial problems are ubiquitous with applications in almost every real-world domain, from planning, scheduling, timetabling, logistics, healthcare to chip designs, Internet data packet routing, etc. Due to this ubiquity, the solving of combinatorial optimization problems has been receiving enormous attention from many scientific communities. This results in a wide variety of solving paradigms, among which are Mathematical Programming, Constraint Programming, Heuristics, Metaheuristics, Satisfiability checking of propositional logic (SAT), Answer Set Programming, Genetic Algorithms, etc. These paradigms can be classified in many different ways. Regarding historical context, they can be classified into “Operations Research” and “Artificial Intelligence”. Regarding the ability to prove the optimality of the obtained solutions, there are exact and heuristics methods. This dissertation focuses on studying complementary paradigms from different categories and how they can be combined with each other.

1.1 Categorizing of solving paradigms for combinatorial optimization problems

1.1.1 Operations Research and Artificial Intelligence

Operations Research (OR) involves studies on how to improve *operations* in practice. OR was started during World War II [1] with the original purpose of optimizing military and industry operations. Traditionally, OR relies heavily on Mathematical Programming. Kantorovich (1939) set an important milestone of combinatorial optimization by introducing the formulation of Linear Programming as a generic problem. An optimization problem is represented as a set of linear (in)equalities, which together form of a *convex polytope* representing the *feasible region* of the problem. In 1947, George Dantzig invented the simplex method (Dantzig, 1987), which essentially iterates vertices of the convex polytope until the vertex of the optimal solution is reached. Linear Programming and the simplex method since then have become one of the foundation bricks of Mathematical Programming and OR in general. The most important extensions of Linear Programming are Integer and Mixed Integer Linear Programming (IP/MIP), which support integer variables. Linear Programming, MIP and their extensions together are generally mentioned as *Mathematical Programming* (MP). Many algorithms and techniques such as Dantzig-Wolfe decomposition, Benders decomposition, Lagrange relaxation have been proposed to handle problems involving large-sized instances. Nevertheless, when the problem size increases, Mathematical Programming tends to fail or become too expensive in memory and time. Heuristics and metaheuristics are then introduced to target complex problems with large-sized instances, by sacrificing optimality for scalability.

Artificial Intelligence (AI), in contrast, was not originally aimed for combinatorial optimization problems. Generally speaking, AI's ultimate goal is to create "intelligence" in machines, or, to be more specific, at making machines perform human's tasks intelligently. Alan Turing's machine, which was used to break the "Enigma" code of Germany during World War II, can be considered as the first intelligent machine. The term Artificial Intelligence was first proposed by John McCarthy in 1956 at the Dartmouth College summer AI conference. At that time, AI was promoted as a tool that could change the world, and hence attracted intense attention from researchers and organizations during the 50s and 60s. But then, due to the lack of computational power, it stepped into the first "AI winter" during the 70s and 80s. In the late 1990s, the new generation of computers with tremendously increased computational power opened a new blooming era of AI, marked by IBM's Deep Blue, the first computer that defeated the world chess champion at the time, Garry Kasparov, in 1997. AI nowadays is a large field which consists of many sub-fields such as reasoning, knowledge

[1] http://www.hsor.org/what_is_or.cfm

representation, planning, learning, natural language processing. AI techniques have been adopted to solve combinatorial optimization problems and become more and more competitive with traditional OR methods. These techniques can be divided into two main groups. The first group roots in Logic Programming (LP), which is one of the earliest subjects of AI. Logic Programming focuses on Knowledge Representation, i.e. how to represent domain knowledge in an expressive yet tractable language for computers. Constraint Programming, Answer Set Programming, Satisfiability Modulo Theories are all considered as descendants of LP. The second group consists of methods imitating natural or social behaviors. Genetic Algorithms, Neural Networks are prominent representatives of this group.

1.1.2 Exact and heuristic methods

Another way to categorize solving methods for combinatorial problems is their ability to prove the optimality of solutions. There are two categories: exact and heuristics methods. *Exact methods* can prove the optimality of a solution or the non-existence of a feasible solution. Therefore they also know when to stop the search. However, in some difficult problems, exact methods are not always scalable, and consequently are not applicable for many practical applications. Among exact methods are Mathematical Programming, Dynamic Programming (classified under OR), Logic Programming, Constraint Programming (classified under AI). *Heuristic* methods, in contrast, do not guarantee the optimality of the obtained solution, neither allows to prove if a feasible solution exists or not. The strength of heuristic methods lies in their ability to find good solutions with a reasonable amount of resources (i.e. memory, time). Heuristics methods therefore are suitable for real-world applications where large-sized instances have to be tackled or short solving time is required. Among heuristic methods are Local Search-based heuristics (classified under OR), Genetic Algorithms, Neural Networks (classified under AI).

1.2 State-of-the-art and research questions

Techniques from each field have their own characteristics, which make them suitable for a particular type of problem or a specific range of instance size. It is compelling that combining techniques from different categories might bring together their complementary characteristics under an unified algorithm/framework. This combination has been exploited in numerous studies. Hybrid methods are on the rise and currently are state-of-the-art of many difficult problems. Traditional exact solvers such as CPLEX, Gurobi all support heuristic/metaheuristic options. Many hybrid systems support different solving paradigms under an unified framework and allow for

combinations between them, e.g. OscaR (OscaR Team, 2012), Minizinc (Nethercote et al., 2007), Essence (Frisch et al., 2007). This trend is also reflected through a series of conferences and workshops specifically devoted to hybrid methods such as CPAIOR - the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (Hooker, 2011) (1999-2018, annual), Hybrid Metaheuristics (Aguilera et al., 2019) (2004-2019, every three years), Matheuristics (Maniezzo and Stützle, 2016) (2006-2018, bi-annual). Many articles written by pioneers from many domains advocate the "marriage" between paradigms from different fields (de la Banda et al., 2014, Gomes, 2000, Hooker, 2007, Swan et al., 2015).

This dissertation centers around the same research topic on the combination of different solving paradigms for combinatorial problems. This combination is exploited at two levels. At a high, general level, we study the relationship between *Knowledge Representation* and *Local Search-based heuristics*, which is in our opinion less-studied and not well-exploited in the literature. At a low, specific level, we study two particular combinatorial optimization problems and how different solving approaches can be applied to them.

The first level is covered in the first part of the manuscript. *Knowledge Representation* (KR) is a traditional, well-studied topic of AI. It investigates how knowledge should be represented and utilized efficiently in problem-solving using computers. Logic Programming (LP), Answer Set Programming (ASP) and Constraint Programming (CP) are AI methods for combinatorial problems which are rooted in KR, hence tend to offer rich representations of knowledge. The trade-off that comes with them, however, is weak efficiency compared to OR solvers. These methods are, to some extent, all based on the idea of *problem solving = problem modelling + solving*. A problem is specified as a *model* using a high level descriptive language. The model is then solved using a black box solver. The clear separation between problem modelling and problem solving allows users to focus on the problem itself instead of algorithmic and technical details. LP and ASP languages stay close to the root of formal languages, while CP languages tend to branch away from their LP origin with many extensions and high level abstractions. The efficiency of AI solvers, although has been improved tremendously in the last decades, is still not sufficient for many real-world problems. In OR, local search-based heuristics is a successful mechanism that is widely used in many real-world applications. Local search-based heuristics are incomplete. They systematically apply local changes (*neighbourhood moves*) to a solution to improve its objective value. Local search heuristics, however, lack in formal, readable representations, which leads to many issues in developing, reusing, maintaining the algorithms. There have been lots of work in literature exploiting this complementary relationship between KR and local search algorithms. AI solvers nowadays almost always offer heuristics options. Some well-known examples are GSAT (Selman et al., 1992), WalkSAT (McAllester et al., 1997, Selman et al., 1994) among SAT solvers, HCLASP (Gebser et al., 2013), WASP (Alviano et al., 2015) among ASP solvers. Most of these, however, are simple

heuristics integrated into their built-in search-tree-based algorithms. Actual local search heuristics modelling are available in a few solvers such as SATenstein (KhudaBukhsh et al., 2009) in SAT or WSAT(CC) (Liu and Truszczyński, 2004) in ASP. However, they only support very simple forms of modelling such as flipping variable's assignments.

In CP, the picture is much more vibrant with "Constraint-Based Local Search" (CBLS), a solving paradigm that was introduced for the first time in Comet (Van Hentenryck and Michel, 2007). In CBLS, a problem is modelled using a CP language and solved by a local search algorithm generated automatically from information extracted from the CP model. Nowadays, many CP and related systems support CBLS as one of the solving options. Some prominent examples include Comet (Fontaine and Michel, 2012, Van Hentenryck and Michel, 2007, 2013), Minizinc (Björdal et al., 2018, Dekker et al., 2018), OscaR (De Landtsheer and Ponsard, 2013, De Landtsheer et al., 2018) and Essence (Akgun et al., 2018). The early works in CBLS mostly derive neighbourhood moves automatically from their data structures and abstractions. Later works however are more flexible and allow users to specify more components of local search algorithms by providing specialized abstractions and facilities. CP languages hence are no longer as "formal" as LP or ASP languages.

However, there remain some issues unexploited by literature. Systems with formal languages such as LP and ASP mostly just integrate local search into their built-in search tree algorithms and hence haven't exploited its full potential strength. Some limited work in actual local search modelling stops at very fundamental levels. CP systems offer a powerful and flexible way to model and exploit local search but in exchange, they lose many potential advantages of formal languages from extending and specializing their languages to target satisfaction and optimization problems. Formal languages are backed up by nearly 70 years of research of Knowledge Representation. They have many advantages and are popular in many application domains such as verification, database, theorem proving, simulation. This link between formal languages and local search heuristics, hence, in our opinion, is a weak point in literature. In the first part of this dissertation, we aim to investigate this link. To be more specific, we target to answer the following questions:

1. Can local search heuristics be specified using a formal, descriptive language?
2. Can the efficiency of a formal language's solver be increased by introducing local search heuristics to the system?

In order to answer these questions, we propose "declarative local search", a framework which allows for modelling local search algorithms declaratively and simulating the algorithm from its descriptions. The research is put in the context of the Knowledge Base System (KBS) IDP (De Cat et al., 2018), which takes FO(·)- an extension of First Order Logic, as the input language. The contribution of the first part is as follows:

- "Declarative local search" functions as a framework where users can specify local search heuristics declaratively. From the OR perspective, the framework helps to achieve a declarative, modular and reusable formalization of local search heuristics.
- From the knowledge representation perspective, we offer an alternative back-end for combinatorial optimization problems in the KBS IDP and hence extends its efficiency in some classes of problem.

In the second part, the combinations between different solving mechanisms are investigated at a low level. We study two specific case studies, namely the Intermittent Travelling Salesman Problem and the Radiotherapy Scheduling Problem.

The Intermittent Travelling Salesman Problem (ITSP) is presented in chapter 4. An exact branch-and-price approach is proposed to tackle small-sized instances, while large-sized instances are tackled using a metaheuristics with sub-problems solved by Linear Programming. The contribution of this chapter is as follows:

- We introduce a new variant of the Travelling Salesman Problem. The problem's characteristics are analysed and a problem library is created and made publicly available for further research.
- We propose a decomposition model including a Mixed Integer Programming and a Linear Programming formulation for the sub-problems. These are integrated in am exact branch-and-bound approach.
- We propose two constructive heuristics and two perturbation strategies, which are integrated into a Variable Neighborhood Search (VNS) framework to tackle large instances.

The Radiotherapy Scheduling Problem (RTSP) is presented in chapter 5. It is a real variant of the RTSP at CHUM, a large cancer center in Montréal, Canada. A two-phase approach is proposed for the problem. For the first phase that assigns patient treatments to specific linacs and days, a Mixed Integer Linear Programming (MIP) model is proposed and solved using CPLEX. For the second phase that decides the sequence of patients on each day/linac and the specific appointment times, a MIP and a Constraint Programming (CP) model are proposed. The models are tested on the dataset generated based on real data provided by CHUM. The contribution of this chapter is as follows:

- Even though the problem is a real-world problem from a particular cancer center, the models are made as general as possible by taking into account many constraints and preferences which were not considered together before in literature.

- To the best of our knowledge, this is the first time Constraint Programming is successfully applied for the RTSP. In the RTSP literature, most work focuses on MIP or heuristics while CP is neglected. We show that CP outperforms the conventional MIP approaches and is able to solve real-sized instances to optimality.
- We describe how realistic instances are generated based on real data. Most works in literature either test their algorithms on empty instances with no fixed appointments or do not mention how their test instances are generated. However, in real life, the planning is usually made on a partially-full schedule which is filled with appointments from the previous planning periods. The schedule's layout and characteristics deeply affect the algorithm's performance. Hence it is important to test the algorithms on close-to-realistic instances.

1.3 Structure of the dissertation

The dissertation is divided into two parts. The first part is devoted to the link between knowledge representation and local search heuristics while the second part presents two particular problems where hybrid approaches are proposed.

The first part, **declarative local search in predicate logic**, is covered in two chapters.

Chapter 2 first introduces the two domains of Knowledge Representation and Local Search-based heuristics. Some background and literature are given and their positions in solving Combinatorial Optimization Problems are analyzed. The links between the two domains are then discussed and the foundation of our work is established accordingly.

Chapter 3 describes *declarative local search*, a framework built on top of the Knowledge Base System IDP, which enables local search heuristics to be synthesized from their formal specifications. It is introduced as a new inference for combinatorial optimization problems in IDP. To model a local search heuristic, users need to specify its components, among which neighbourhood moves are the most important. Two types of neighbourhood moves, namely standard moves and Large Neighbourhood Search moves, are supported. A set of built-in local search heuristics is provided, enabling users to combine neighbourhoods in different ways. The new local search inference is then demonstrated to be able to complement the existing solving mechanisms of IDP.

The content of this chapter has been published in two articles:

- Tu-San Pham, Jo Devriendt, and Patrick De Causmaecker. Declarative local search for predicate logic. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 340–346. Springer, 2019a.

- Tu-San Pham, Jo Devriendt, and Patrick De Causmaecker. Modelling Local Search in a Knowledge Base System. In *New Trends in Emerging Complex Real Life Problems*, pages 415–423. Springer, 2018a.

The next two chapters belong to part two, **hybrid algorithms for specific case studies**.

Chapter 4 introduces the *Intermittent Travelling Salesman Problem* (ITSP), a new variant of the Travelling Salesman Problem. The ITSP is inspired by real world drilling/texturing applications. In this problem, each vertex can be visited more than once and there is a temperature constraint enforcing a time lapse between two consecutive visits. A branch-and-bound approach is proposed to solve small instances to optimality while four Variable Neighbourhood Search metaheuristics are proposed to produce high quality solutions for large instances. An instance library is built and made publicly available. The content of this chapter has been published in:

- Tu-San Pham, Pieter Leyman, and Patrick De Causmaecker. The intermittent travelling salesman problem. *International Transactions in Operational Research*, pages 525–548, 2018b.

Chapter 5 introduces the *Radiotherapy Scheduling Problem* (RTSP). Radiotherapy treatment is a popular form of cancer treatment where a patient receives a large amount of radiation over a course of several consecutive days. In the most common form of radiotherapy, the radiation is delivered by a linear accelerator (linac). The RTSP consists of finding the treatment schedule for a set of patients given a set of linacs respecting many side constraints and preferences. In this chapter, a Mixed Integer Linear Programming MIP formulation is given for the problem. Since the formulation is not tractable, a two phase approach is proposed. In the first phase, patient treatments are assigned to specific linacs and days. The second phase then decides the sequence of patients on each day/linac and the specific appointment times. For the first phase, a MIP model is proposed and solved using CPLEX. For the second phase, a MIP and a Constraint Programming (CP) model are proposed. The test data is generated based on real data from the cancer center of CHUM, a large cancer center in Montréal, Canada, with about 3500 new patients and 40.000 treatments a year. The results show that CP approach outperforms MIP in this problem and is able to solve real-sized instances. The content of this chapter has been accepted for publication in:

- Tu-San Pham, Louis-Martin Rousseau, and Patrick De Causmaecker. The Radiotherapy Scheduling Problem. In *Proceedings of MISTA'2019*, 2019b.

In summary, the center focus of this dissertation is in describing and solving combinatorial problems. Solving a combinatorial problem should merely focus on the problem itself. Therefore narrowing the gaps between different paradigms regardless of

their historical origins or philosophies is necessary to unify all available resources for the final goal of solving a problem at hand. The two parts of this dissertation together contribute to this research theme by exploiting the relationships between different solving paradigms at two complementary levels. We consider this dissertation as the first steps on a long and promising boulevard ahead.

Part I

Declarative local search in predicate logic

Chapter 2

Knowledge Representation and Local Search-based heuristics

Goal of the chapter Knowledge Representation belongs to Artificial Intelligence while Local Search is a solving paradigm from Operations Research. This chapter gives the background and literature of Knowledge Representation and Local Search, analyses their positions in the solving of combinatorial optimization problems. We then discuss the links between the two domains and situate our work on the bridge between them.

2.1 Knowledge Representation and Knowledge Base System

2.1.1 Knowledge Representation

Knowledge is fact, information, understanding of human about the world, which is accumulated through observation, activities or passed from generation to generation. In everyday life, we store, process and *reason* knowledge to solve problems and tasks. For example, in the hunter-gatherer time, we need knowledge about which fruit is edible, where to find water, how to bring down a big animal in order to survive. In the current time, we need knowledge about money, tax, car driving, etc., to solve daily

tasks. *Knowledge* is hence vital in problem-solving. The ultimate goal of Artificial Intelligence is to make computers perform problem-solving. For this purpose, we need to provide computers with sufficient knowledge about problem domains. *Knowledge Representation* (KR) is the study of how to represent knowledge in a machine-readable form and how to utilize it in problem-solving using computers. KR is one of the most important sub-areas of AI.

From the early days of AI, the late 1960s and early 1970s, there were some debates on the two paradigms in the representation of knowledge: declarative versus procedural. Advocates of declarative representations, mostly based in Europe, believed that logic could bring important benefits in solving computational problems compared to standard programming languages. In 1958, John McCarthy proposed *Advice Taker*, a hypothetical computer program, which is the first proposal to represent information as logic for a program. In 1974, Robert Kowalski's paper *Predicate Logic as a Programming Language* (Kowalski, 1974) proposed to use first-order predicate logic as a programming language, which laid the foundations for **Logic Programming** (LP). Since then, LP has diversified into many branches with various variants and extensions. Those variants can be classified into two main paradigms: *theorem proving-based* and *model generation-based*. Theorem proving-based uses *deduction* as an inference method, i.e. a solution is given by a derivation of a query. Prolog (Colmerauer, 1990) is the most prominent representative of this paradigm. Model generation-based uses *model generation* as an inference method, i.e. a solution is given by a model of the problem representation. Propositional satisfiability (SAT), Constraint Programming (CP) and Answer Set Programming (ASP) belong to this paradigm.

SAT solving technologies have been very well studied and achieved tremendous progress in the last decades. However, its language, which is restricted to propositional logic, is not suited as a language for developing models. ASP and CP languages, on one hand, are more expressive with high-level propositions and abstractions allowing for easy modelling of practical domains. On the other hand, they inherit rich solving techniques from SAT. Some prominent CP systems are Zinc (Marriott et al., 2008), OPL (Van Hentenryck et al., 1999), ESSENCE (Frisch et al., 2008), Comet (Michel and Van Hentenryck, 2005) and Gecode (Gecode Team, 2013) while DLV (Leone et al., 2006), clasp (Gebser et al., 2007a), Smodels (Niemelä et al., 2000), Gringo (Gebser et al., 2007b) are among successful ASP solvers.

High-level, expressive languages and powerful solvers make CP and ASP the best candidates among logic programming techniques for Combinatorial Optimization Problems solving. There is a rich literature on the use of CP and ASP in COPs solving. Regarding CP, Govindarajan et al. (1994) and Guo and Jayaraman (2008) present Preference Logic Programming, an extension of CP, as an inference to solve optimization problems. In (Toussaint, 2015), the sequential robot manipulation is formulated and solved as a first-order logic extension of a mathematical program. Le et al. (2017) present how to model distributed constraint optimisation problems as

logic programs and propose an algorithm based on logic programming. ASP examples include, among many others, the solving of a combinatorial auction problem (Baral and Uyan, 2001), a team-building problem at a seaport (Ricca et al., 2012), a nurse scheduling problem (Alviano et al., 2017), personnel scheduling at a seaport (Ricca et al., 2012) or a supertree construction problem (Koponen et al., 2015).

2.1.2 Knowledge Base System

Knowledge Base System is a paradigm born from the idea of separating knowledge from computation. In real life, knowledge of a problem domain can be used to solve various problems and tasks. For example, a university course scheduling application should be able to solve many related tasks such as generating schedules at the start of the year; verifying the correctness of hand-made and revised schedules; generating a reason of why a certain schedule is not consistent; retrieving the schedule of a student or a professor, etc. Those tasks are different in nature but all based on the same domain knowledge, e.g. professors teaching courses, students studying, resources available and rules to be respected. Based on this idea, KBS paradigm is proposed to allow for expressing knowledge of a problem domain in a declarative language and solving different computational tasks around this resulting *knowledge base*. A KBS usually consists of two main components: (i) a formal declarative language that allows describing domain knowledge; and (ii) a set of inference methods to solve a wide variety of tasks around that knowledge base.

The IDP (*Imperative-Declarative Programming* (De Cat et al., 2018)) system is an experiment to build such a KBS, using FO(·) (De Cat et al., 2018) as the input language. FO(·) is based on classical first-order logic (FO). It extends FO with inductive definitions, types, aggregates, and arithmetics. FO(·) is closely related to ASP, but adopts a more traditional view of classical logic. IDP supports a procedural interface through the language Lua. It is equipped with a rich set of inference methods that allow solving different kinds of tasks. For example, for the aforementioned course scheduling application, IDP can generate schedules at the beginning of the semester using the *model expansion* or *optimization* inference; verify the correctness of schedules using the *model checking* inference; generate a reason of why a certain schedule is not consistent using the *explain unsat* inference; proving invariants on the knowledge using the *theorem proving* inference – all using the same knowledge base. IDP has been used in many application domains such as interactive configuration (Van Hertum et al., 2017), machine learning, data mining (Bruynooghe et al., 2015) and access control systems (Cramer et al., 2014).

However, as an inevitable trade-off between expressiveness and efficiency, solving big combinatorial optimization problems still remains a weak-point of IDP. Currently, to solve combinatorial optimization problems, IDP employs a two-phase ground-

and-solve approach, similar to that of ASP. It is well-known that grounding is the bottleneck of ASP (Balduccini et al., 2013) due to the exponential size grounding in general. Therefore, IDP faces the same problem with ASP in problems with large-sized instances. It is also the weakness of other logic-based systems.

2.2 Combinatorial optimization problems and local search-based heuristics

Combinatorial optimization has been attracting great attention from researchers over the last decades, which results in a wide range of solving paradigms that is still growing relentlessly. Recent years have witnessed tremendous progress of exact methods such as Mathematical Programming or Constraint Programming. However, they still face real challenges when it comes to solving real-world problems, which typically involve large-sized instances and/or require fast solving times. Heuristic methods hence still play an important role in the solving of combinatorial problems and attract great attention from operations researchers. Among them, local search based heuristics represent a very successful branch.

Local search is an heuristic search method which improves the objective value of a solution by iteratively applying some local changes, which are known as *neighbourhood moves*. Formally, a neighbourhood move is a function which maps a solution to another, which is addressed as its *neighbouring solution*. Given an initial solution, a local search algorithm explores the search space by enumerating the solution's neighbours. Once a neighbouring solution is accepted according to some acceptance criteria, e.g. if it improves the objective value of the current solution, it becomes the starting point of the new iteration. The algorithm terminates when a stopping criterion is met, the solution with the best objective value is then returned. This solution is the *local optimum* regarding the neighbourhood. A local optimum, however, can be quite far away from optimality. Therefore, some *diversification* techniques need to be applied to guide the search to the global optimum. Algorithms that utilize diversification mechanisms to guide some subordinate heuristics to escape from local optima are called *metaheuristics*. There are various ways in which metaheuristics can diversify the search. Some metaheuristics use *non-monotonic search strategies*, which accept worse or non-improved solutions such as Simulated Annealing (Kirkpatrick et al., 1983) or Late Acceptance Hill Climbing (Burke and Bykov, 2012). In the same category, Tabu Search (Glover and Laguna, 1998) prevents revisiting solutions by forbidding a list of recent visits (tabu). Observing that local search depends heavily on the initial solutions, some metaheuristics utilize a multi-start scheme such as Adaptive Multi-start (Boese et al., 1994), GRASP (Feo and Resende, 1995). Some other metaheuristics alternate between different neighbourhoods like Variable Depth Search (Lin and Kernighan,

1973) or Variable Neighbourhood Search (Hansen and Mladenović, 1997) (VNS). In another approach, Iterated Local Search (Lourenço et al., 2003) (ILS) allows simple (meta)heuristics to complement each other by combining them as subordinate procedures in an iterated framework.

At any time of the search, local search-based heuristics consider only one solution (hence another name "single-point iterative search" or "trajectory search"). This characteristic distinguishes local search based heuristics from population-based heuristics which consider a set (population) of solutions at the same time. Prominent representatives of population-based approaches include Genetic Algorithms (Koza and Koza, 1992), Evolutionary Programming (Fogel, 1994), Scatter Search (Glover et al., 2000), Particle Swarm Optimization (Eberhart and Kennedy, 1995) or Ant Colony Optimization (Drigo, 1996). Most of those approaches are motivated by evolution or social behavior. In practice, OR practitioners usually combine ideas from the above methods to create their own approaches and this kind of creativity in solving problems pays off. Many successful algorithms are hybrid which combine algorithms from different categories.

From this point, we use the term *local search* to imply local search-based algorithms, which include both heuristics and metaheuristics.

2.3 Motivation: the bridge between Declarative Knowledge Representation and Local Search heuristics

In this section, we discuss the links between Declarative Knowledge Representation and Local Search in the solving of Combinatorial Optimization Problems and locate our research on the bridge between them. We firstly analyse the strengths and weaknesses of KR and LS, and then discuss how they can complement each other.

Knowledge Representation The strength of declarative KR approaches, or of Knowledge Base Systems (KBS)) in particular, is the separation between problem-modelling and problem-solving. Problem modelling is done using an expressive descriptive language while the solving is handled by a black box solver. This separation brings many advantages to operations research practitioners.

First, most modelling languages are based on logic, which is historically well-known as an expressive language. New generations of descriptive languages such as ASP and CP languages are facilitated with high-level abstractions, which make it even easier

to represent real-world problems. This allows domain experts and OR practitioners to focus on the problem itself instead of techniques and implementation details.

Second, this separation allows for easily tackling changes in requirements or adapting to different variants of a problem. In most of the cases, small changes can be handled by modifying the modelling without worrying about the underlying computational methods. The descriptive modelling is, to some extent, accessible to domain experts and hence requires less intervention from programming experts.

Third, due to the compactness of programs in declarative languages, it is usually easier to isolate issues compared to programs in conventional procedural languages. The program is hence less error-prone. For this reason, most of the high-precision applications make use of declarative languages, e.g. Prolog is used in a decision Support System for the Space Shuttle in (Nogueira et al., 2001); CHIP, an extended Prolog, is used in a Fault Diagnosis in Digital Circuits (Simonis and Dincbas, 1987); A-Prolog is used in a diagnostic reasoning system (Balduccini and Gelfond, 2003).

Finally, standing on the shoulder of the giant Classical Logic, KR systems inherit a rich resource of facilities. Many inference methods are available, which support the building of complete, self-contained applications with the ability to solve a wide range of tasks and problems. They also benefit from the fast-growing automated reasoning technologies. The computational power of the solvers hence keeps improving every year.

However, the disadvantages of those systems are also indisputable. The last decades have witnessed tremendous improvement in black-box solvers technology, especially SAT and CP. However, compared to other solving approaches, there still remains lots of room for improvement. If no sophisticated procedures are specified, those solvers still face real challenges in solving real-world large-sized problems with a practical time budget.

Local search heuristics The most pronounced strength of local search is the ability to achieve good solutions for large-sized problems with short solving times. This makes local search approaches particularly suitable for some important classes of problem such as scheduling or planning. However, in literature, local search algorithms are mostly procedural-based, i.e. they are presented in terms of low-level, algorithmic concepts. Modularity and reuse are therefore difficult to obtain. As a consequence, if the program is not well-designed, it is difficult to adapt to a new problem or to take into account the change of requirements. Even in a well-designed program, it often requires lots of intervention from program experts to do so. It indicates the lack of expressiveness and abstractions of local search currently. This issue is also exposed through the usually considerable distance between the specification and the final program.

This weakness of local search heuristics has been identified and tackled by many operations researchers. Some frameworks such as HotFrame (Fink and Voß, 2003) and EasyLocal++ (Di Gaspero and Schaerf, 2003) provide C++ libraries with design patterns, abstract classes and work as a foundation to build local search algorithms. Those frameworks however are still procedural and programming knowledge demanding, which does not solve most of the issues listed above. Hyper-heuristics (Burke et al., 2003) operate on a higher level of abstraction. A hyper-heuristic allows automated selecting and/or generating heuristics from a set of well-designed, low-level heuristics. Hyper-heuristics are hence described as "knowledge poor" heuristics, i.e. they are non-problem specific and can be developed and deployed with little knowledge of the problem domain. The set of low-level heuristics are crucial in hyper-heuristics but are not fully available for academic research. More overview of hyper-heuristics can be found in (Burke et al., 2013, Özcan et al., 2008).

The remarks above make it obvious that declarative knowledge representation and local search are complementary to each other and hence, should be combined together. This topic actually has been discussed quite often in literature. Hooker (2007) pointed out that the CP and OR communities have focused too much on solution methods while neglecting modelling. He strongly argued that "modeling is what a scientific discipline called CP or OR should be all about" and suggested that a good future for CP is the **prescriptive modeling** of human activities instead of focusing on computational methods. Swan et al. (2015) advocated a standardized, explicit, formal description of metaheuristics. The authors listed lots of benefits such a description can bring up, including communicability, reproducibility, interoperability, automated assembly of metaheuristics and efficiency. Similarly, de la Banda et al. (2014) pointed out one of the challenges in optimization technology is the design of high-level modeling languages. Align with this ideology, we propose "declarative local search" and locate our work on the bridge between KR and local search.

2.4 Declarative local search

"Declarative local search" is an experiment to build a declarative framework for local search approaches, where users can specify problem specifications and building blocks of local search declaratively using a high-level formal language. Those descriptions are then taken as inputs by a solver to synthesize local search heuristics to solve the problem. The framework is built on top of the Knowledge Base System (KBS) IDP with FO($\cdot$) as the input language. We hereby argue about the motivations of our work.

From the operations research perspective, declarative local search offers the following benefits. First, by modelling local search declaratively, we hope to achieve a declarative, modular and reusable formalization of local search heuristics. Second, modelling local

search algorithms makes them less error-prone and more maintainable. Descriptive modelling is fairly natural, human-readable and compact. It is hence easier to detect any issue in the description compared to a procedural description, which is typically encoded in thousands of lines of code. Third, it reduces time and effort taken to develop algorithms, which helps programmers and domain experts focus more on the problem itself rather than on computational techniques. In addition, it can also serve as a fast prototype tool to experiment and analyse local search heuristics. Fourth, by using a logic-based language, declarative local search benefits from existing inference methods and facilities from the logic community. For example, in our approach, initial solutions are generated automatically using *model expansion*, a built-in inference of IDP. It is especially helpful in tightly-constrained problems whose feasible solutions are difficult to obtain. The neighbourhoods can be debugged using the *explainunsat* inference (which gives the reason why a solution is not feasible). Fifth, since the framework makes use of existing automatic reasoning techniques such as query solving or SAT solving, its computational power also benefits from the progress of those techniques.

From the knowledge representation perspective, declarative local search offers an alternative back-end to solve optimization problems. The ultimate goal of a knowledge base system is to tackle all types of problems efficiently - including those for which the current state-of-the-art employs local search. Our work transfers a successful solving approach from the operations research community to declarative solving, which could serve as an alternative or complement to the existing solving mechanisms for logic programming. Although the framework is built on top of IDP, the idea is general and can be applied to any other declarative solver.

2.5 Related work

We divide related work into two categories. The first one is local search in Logic Programming while the second one is declarative modelling of local search algorithms.

2.5.1 Local search in Logic Programming

Most logic programming algorithms are exact methods, so it is necessary to incorporate heuristics/local search to solve hard problems. We hereby discuss heuristics/local search algorithms in two prominent Logic Programming paradigms: SAT and ASP.

Local search in SAT solving SAT solvers can be categorized into two groups: (1) back-tracking search-tree based algorithms; and (2) stochastic local search-based algorithms. Algorithms from the first group are exact methods, among which

Davis–Putnam–Logemann–Loveland (DPLL) algorithm (Davis et al., 1962) is one of the first and most famous ones. Modern SAT solvers are mostly based on Conflict-Driven Clause Learning (CDCL) (Marques-Silva and Sakallah, 1999), which is inspired by DPLL. Algorithms from the second group are heuristic methods. Local search for SAT was first proposed independently by Gu (1992) and Selman et al. (1992). Among many local search algorithms for SAT, the most well-known are GSAT (Selman et al., 1992) and WalkSAT (McAllester et al., 1997, Selman et al., 1994). They all work on the same principle: starting from an initial complete solution, they iteratively "flip" a (boolean) variable assignment, which is equivalent to a "neighbourhood move" in a conventional local search algorithm, until all the clauses are satisfied. The mechanism that variables are selected to flip, nevertheless, distinguishes algorithms from each other. GSAT tries to minimize the number of unsatisfied clauses in the new assignment in a descent manner with some randomness, while WalkSAT flips a variable within a clause that is unsatisfied by the current assignment and breaks the tie randomly. More review on local search for SAT can be found in (Hoos and Stützle, 2000).

Many hybrid approaches have been proposed to combine the strength of search-tree based and local search approaches. For example, in (Mazure et al., 1998) and (Habet et al., 2002), a local search procedure is utilized to assist the variable selection procedure of the DPLL. A tighter combination between local search and SAT DPLL solver in an incremental framework is proposed in **HBISAT** (HyBrid Incremental SAT Solver) (Fang and Hsiao, 2007). HBISAT invokes local search to retrieve a set of clauses which will be passed to a DPLL solver, which in turn pass the output back to the local search solver to improve its output.

Regarding modelling of local search in SAT, **SATenstein** (KhudaBukhsh et al., 2009) is the only SAT system supporting this facility. SATenstein allows for synthesizing local search SAT solvers by combining existing algorithmic components. However, the framework strictly works on binary assignment of boolean variables and does not allow for modelling new "neighbourhood moves" or components.

Local search in ASP Computation methods in ASP are mostly based on the backtracking search tree-based algorithm CDCL (Marques-Silva and Sakallah, 1999), which was initially introduced for SAT solving. Most heuristic methods for ASP involve altering the variable selection/branching/backtracking in the search tree. In (Gebser et al., 2013), the ASP's input language is extended with some language elements to express domain-specific heuristics. Their framework **HCLASP**, built on top of the ASP solver *clasp*, provides a flexible tool for the users to manipulate the heuristics in the core CDCL solver. Similarly, in (Dodaro et al., 2016), the ASP solver WASP (Alviano et al., 2015) is extended with an interface which allows for specifying domain-specific heuristics declaratively in ASP. Those declarative heuristics are utilized to modify/guide the built-in heuristics of the ASP solving. Nonetheless, both (Dodaro et al., 2016) and

(Gebser et al., 2013) support only search tree-based heuristics, not local search. To the best of our knowledge, **WSAT(CC)** (Liu and Truszczyński, 2004) is the only local search solver in ASP. WSAT(CC) operates by flipping variables assignments to find a better solution. It is, however, a very simple solver with only a couple of options on how to choose variables to flip.

2.5.2 Local search in Constraint Programming

Constraint Programming, historically rooted in Logic Programming, is a successful paradigm with expressive languages and powerful solving mechanisms. In this section, we give an overview of Local Search modelling in Constraint Programming.

The first attempt in formalizing local search heuristics in CP can be dated back to 1997 when **Localizer** (Michel and Van Hentenryck, 1999) introduces the first modelling language for local search with the use of invariant to specify incremental algorithms declaratively. Its successor **Comet** (Van Hentenryck and Michel, 2007) is the first system that introduces the "Constraint-based Local Search" (CBLS) paradigm. CBLS is based on the idea $Localsearch = Model + Search$. To solve a problem in Comet, two components need to be specified: (1) a high-level model describing the problem; and (2) a *procedural* abstraction search procedure to exploit the problem structures. Since Comet, CBLS has gained popularity and is supported widely in many systems. **CML** (the Comet Modelling Language) (Fontaine and Michel, 2012), the direct descendant of Comet, introduces abstract models and model operators to allow for specifying and manipulating solver independent models in Comet. **Objective-CP** (Van Hentenryck and Michel, 2013), the latest descendant in the Comet family, supports parallelization of multi-start and branch and bound algorithms.

OscaR (OscaR Team, 2012), a Scala toolkit for solving Operations Research problems, also offers a CBLS module, namely **OscaR.cbls** (De Landtsheer and Ponsard, 2013). OscaR.cbls provides many built-in general-purpose neighbourhoods which are often used in combinatorial problems, e.g. *assignNeighborhood* that changes the value of a variable, or *swapNeighborhood* that swaps the values of two variables. OscaR.cbls also supports some **search combinators** (De Landtsheer et al., 2018) which allow for specifying the combination and exploration of neighbourhoods. New neighbourhoods can also be defined *procedurally* using a common interface, but not declaratively.

Another prominent family of CBLS is the Minizinc. In 2007, MiniZinc (Nethercote et al., 2007) was introduced in the G12 platform with the ambition to become a standard CP modelling language. Since then, many connecting languages/frameworks have been built around MiniZinc which support Local Search modelling. **Search combinators** (Schrijvers et al., 2011) is a combinator language which allows users to interact with the search at every node of the tree, hence essentially replaces the solver's

Comet	Localizer (Michel and Van Hentenryck, 1999)	1999
	COMET (Van Hentenryck and Michel, 2007) (successor of Localizer)	2007
	Comet Modelling Language (CML) (Fontaine and Michel, 2012)	2012
	Objective-CP (Van Hentenryck and Michel, 2013) (successor of CML)	2013
Minizinc	Search Combinator (Schrijvers et al., 2011)	2013
	Minisearch (Rendl et al., 2015) (successor of Search Combinator)	2015
	fzn-oscar-cbls(Björdal et al., 2015)	2015
	Declarative Local-Search Neighbourhood in Minizinc (Björdal et al., 2018)	2018
	Solver-independent Large Neighbourhood Search (Dekker et al., 2018)	2018
Kangaroo	Kangaroo (Newton et al., 2011) CBLS system using lazy propagation	2011
OscaR	OscaR.cbls (De Landtsheer and Ponsard, 2013)	2013
	Search combinator for OscaR.cbls (De Landtsheer et al., 2018)	2018
Essence	Structured Neighbourhood Search for Essence (Akgun et al., 2018)	2018

Table 2.1: CBLS systems.

built-in search. Its descendent **MiniSearch** (Rendl et al., 2015) is more restricted, hence also more light-weight. **MiniSearch** only interacts with the solver's search at the granularity of solutions. It also supports Large Neighbourhood Search (LNS) and allows for intervention in the search procedure to a certain extent. **fzn-oscar-cbls** (Björdal et al., 2015) is a CBLS backend for Minizinc, which essentially translates a Minizinc model to an OscaR.cbls model and solves it by calling OscaR.cbls as a black-box solver. The most current work in the family of CBLS in Minizinc, **fzn-oscar-cbls**'s direct descendent (Björdal et al., 2018), allows fully modelling local search neighbourhoods declaratively by extending the Minizinc syntax. The user-defined neighbourhoods are then integrated into the search procedure alongside with the automatic built-in neighbourhoods. The modelling of meaningful neighbourhood moves in LNS in Minizinc is supported in Solver-Independent Large Neighbourhood Search (Dekker et al., 2018).

Essence (Frisch et al., 2007) is another CP system with strong CBLS support. Their CBLS framework (Akgun et al., 2018), namely Structured Neighbourhood Search (SNS), derives neighbourhood structures automatically by exploiting abstract structures in the problem's specifications and performs a built-in local search on the derived neighbourhoods. Regarding the implementation-level, each neighbourhood is expressed as a Constraint Optimization Problem and solved with a constraint solver. Essence is similar to our work in terms of the usage of an exact solver to solve sub-problems to generate neighbouring solutions. However, it is based on a high-level CP language and automatically derives neighbourhoods based on pre-defined moves on those abstractions.

Kangaroo (Newton et al., 2011) is a light-weight CBLS system using lazy propagation, which focuses more on efficiency and memory consumption. Kangaroo is provided as a set C++ libraries, hence, does not offer declarative modelling. A summary of CBLS systems can be found in table 2.1.

2.5.3 Other paradigms

Besides Local search in Logic Programming and Constraint Programming, there are other works devoted to formalizing local search or at least for easing the hardship of building a local search algorithm. **Localsolver** (Benoist et al., 2011) introduces the idea of "local search programming" – a model-and-run paradigm for local search which combines a simple modelling grammar and an incremental solver. Its language **LSP** ("Local Search Programming") is close to classical mathematical programming formalisms, extended with common mathematical operators. Their neighbourhood moves use variable-flip and ejection chain-like algorithms.

The **Search-inter-relax SIMPL** (Aron et al., 2004) is a general-purpose solver which provides a set of constraints which enables users to combine different technologies like MILP, CP into one hybrid-algorithm. SIMPL supports local search, however, in a restricted way. It solves problems by enumerating problem restrictions (by adding more constraints) and they see local search neighbourhood as the feasible set of a problem restriction. SIMPL does not support user-defined neighbourhoods or deep intervention in local search algorithms.

Other frameworks such as **HotFrame** (Fink and Voß, 2003) or **EasyLocal++** (Di Gaspero and Schaerf, 2003) are provided as C++ libraries with design patterns and abstract classes to ease the effort of building local search algorithms. However, they are barely declarative.

2.5.4 Discussion

Even though a pure model-and-run is the final ideal system, domain knowledge and OR expert knowledge still play a major role in the efficiency of the approach in many problems. This is reflected through lots of effort in modelling domain and OR knowledge declaratively on top of automatic black-box algorithms. Sharing the same target, our "declarative local search" enables declarative modelling of user-defined neighbourhoods and hence allows problem-specific knowledge to be integrated into the system easily. The main difference is the context of our work. To the best of our knowledge, this is the first time local search neighbourhoods can be modelled declaratively in a logic context. All CBLS systems work around their own languages, which are mostly based on CP or Mathematical Programming Languages, enriched with abstract structures. Their solvers are built around those high-level abstractions and are purely devoted to combinatorial optimization problem solving. In contrast, our framework is placed in the context of a KBS and based on a declarative language which is rooted in the famous classical First Order Logic. Since our formalization can be interpreted by an automatic reasoning system, we benefit from its existing

inferences and formal methods as analyzed in section 2.4. In addition, we also provide an alternative back-end for solving optimization problems in a KBS.

Another important benefit offered by our framework is the presence of different solving approaches in the same system, using the same knowledge base. Currently, IDP uses different solving mechanisms to solve different inferences, e.g. MINISAT(ID) for feasibility search and minimization, XSB for query evaluation and Spass for theorem proving. This diversity opens up the potential of combining different methods to solve problems adaptively and efficiently. For example, observing that many combinatorial problems contain *symmetry information* which can be detected automatically from their formal specifications, Devriendt et al. (2015) propose to exploit these information to derive neighbourhood moves for local search algorithms.

Chapter 3

Declarative Local Search in IDP

Goal of the chapter In this chapter we introduce "declarative local search" - a framework built on top of the Knowledge Base System IDP, which enables local search heuristics to be synthesized from their formal descriptions. It is introduced as a new inference for combinatorial optimization problem solving in IDP. To model a local search heuristic, users need to specify its components, among which neighbourhood moves are the most important. Two types of neighbourhood moves, namely standard moves and Large Neighbourhood Search moves, are supported. A set of built-in local search heuristics are provided, enabling users to combine neighbourhoods in different ways. We demonstrate how the new local search inference can be used to complement the existing solving mechanisms of logic programming.

The content of this chapter is based on two publications:

- Tu-San Pham, Jo Devriendt, and Patrick De Causmaecker. Declarative local search for predicate logic. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 340–346. Springer, 2019a.
- Tu-San Pham, Jo Devriendt, and Patrick De Causmaecker. Modelling Local Search in a Knowledge Base System. In *New Trends in Emerging Complex Real Life Problems*, pages 415–423. Springer, 2018a.

I am responsible for the main content while Jo Devriendt assisted with technical aspects of IDP.

3.1 Introduction

IDP (*Imperative-Declarative Programming*) (De Cat et al., 2018) is a Knowledge Base System (KBS) which consists of two main components: (i) a formal declarative language to describe domain knowledge as a *knowledge base*; and (ii) a set of inference methods for solving a wide variety of tasks around a knowledge base. Its language FO(·) is based on classical first-order logic (FO), extended with inductive definitions, types, aggregates and arithmetics. In this work we focus on IDP's ability to solve combinatorial optimization problems, which is provided through the inference method *optimization* using MINISAT(ID) (De Cat et al., 2013b) as the back-end engine. As a CP-SAT-based solver, it shows limited performance on many optimization problems, such as the assignment problem (Devriendt, 2017), or real-world problems with large-sized instances.

We introduce *declarative local search*, a framework for specifying local search heuristics declaratively in IDP. Local search is provided as a new inference method, serving as an alternative to solve optimization problems. Conventionally, to solve an optimization problem in IDP, users need to define a problem's characteristics as FO(·) constraints and an objective term which represents the objective function to be minimized. The problem is then solved using the *optimization* inference, which takes the problem's specifications, the objective term and a specific instance as input. To use the local search inference, besides a problem's modelling, users need to specify necessary components of a local search heuristic, chief of which are the neighbourhood moves. Users can specify multiple neighbourhoods and combine them in different ways using a set of built-in heuristics and metaheuristics. Two types of moves are supported, namely standard move and Large Neighbourhood Search move.

The framework is illustrated using a classical optimization problem – the Travelling Salesman Problem (TSP). Classical neighbourhood moves for the TSP are modelled and used to construct several local search heuristics. The efficiency of this approach is compared to two declarative black box solvers: *clingo* (Gebser et al., 2014) and IDP.

The rest of the chapter is organized as follows. Section 3.2 introduces IDP, its language FO(·) and illustrates how an optimization problem is modelled and solved in IDP. Section 3.3 explains how to model a local search heuristic using the framework while the implementation of the framework is presented in section 3.4. Section 3.5 presents numerical results. In section 3.6, we discuss the limitation as well as the potential of the framework. Finally, section 3.7 concludes the chapter with summary and future work.

3.2 IDP and FO(·)

The IDP system (De Cat et al., 2018) is a Knowledge Base System (KBS) equipped with: (1) FO(·), a high-level language which allows users to specify knowledge of a problem domain; and (2) a set of inferences to solve a wide range of problems around this knowledge. IDP's language, FO(·), is based on classical first-order logic (FO), extended with types, aggregates, arithmetic, inductive definitions, bounded arithmetic, and partial functions. IDP supports many types of inferences: optimization, query solving, explaining unsat, progression, etc. Many of IDP 's inferences are fundamentally different (e.g. theorem proving and model checking), therefore a specific solving approach is applied for each inference method.

For *model expansion* and its variant *optimization* - inference methods to solve combinatorial optimization problems, IDP employs a two-phase *ground-and-solve* strategy where a FO(·) specification is grounded to a set of constraints in Extended Conjunctive Normal Form (ECNF), which will be handled by MINISAT(ID) (De Cat et al., 2013a), a CP-SAT-based solver. A thorough introduction to IDP and FO(·), including examples and demos, can be found at dtai.cs.kuleuven.be/software/idp.

To reason about a problem $\mathcal{P}$, IDP requires an FO(·) model that consists of a number of logical components, namely vocabularies, structures, terms, and theories. A *vocabulary* is a set of symbols, i.e. types, constants, functions and predicates, to represent the problem. A *structure* I over a vocabulary Σ (denoted as $I|_\Sigma$) consists of a *domain* $dom(I)$ and an assignment of a value s^I to each symbol $s \in \Sigma$. We say that I is an *interpretation* of Σ if I assigns values to all symbols in Σ. If I is a structure of vocabulary Σ, and Σ' is a subset of Σ, then $I|_{\Sigma'}$ is the restriction of I to symbols in Σ', or a partial interpretation of Σ. A structure S interpreting all symbols corresponding to the input data of $\mathcal{P}$ is an *instance* of $\mathcal{P}$. The problem's constraints are represented by FO(·) formulas and definitions, which together amount to a *theory*. A theory T must be built over a particular vocabulary *V*, denoted as $T|_V$. Given a partial structure $\bar{S}|_V$ (an instance) and a theory $T|_V$ (the constraints), IDP's inference *model expansion* extends $\bar{S}$ to a two-valued structure S that fully interprets all symbols in V and satisfies all constraints and definitions specified in T. S is then called a *model* of T, representing a solution of $\mathcal{P}$.

To solve an optimization problem on $\mathcal{P}$, a *term* needs to be specified to represent the objective function to be optimized. Given the term obj, IDP's inference *optimization* (*minimization* or *maximization*) is then utilized to find a solution with the best value to obj. The *optimization* inference is a generalization of *model expansion*, implemented as a series of iterated *model expansion* calls, where a new constraint forcing the upper/lower bound of the objective term is added to the model at each iteration. Also, users can specify *queries* and apply the *querying* inference to retrieve information from a model.

As a running example, we employ the Travelling Salesman Problem (TSP). Given a set of nodes and a distance function between them, the TSP consists of finding the shortest Hamiltonian cycle – the *tour* – visiting all nodes. A model for the TSP in IDP can be found at goo.gl/TTv85c.

Example 3.2.1 (TSP)**.** The four components for the TSP modelling are as follows:

- The vocabulary V specifying the parameters ($Node$, $Distance$, $Depot$), whose values define a problem instance, and the decision variables ($Path$, $Reachable$), whose values define a solution of the problem.
- The theory $T|_V$, which is a conjunction of FO($\cdot$) formulas specifying the constraints of the problem:

$$\forall x\colon \exists! y\colon Path(x,y).$$

$$\forall x\colon \exists! y\colon Path(y,x).$$

$$\{\quad Reachable(Depot).$$

$$\quad Reachable(x) \leftarrow \exists y\colon Reachable(y) \wedge Path(y,x).\}$$

$$\forall x\colon Reachable(x).$$

  The first two lines of the theory in the example represent the flow constraints. Line three and four amount to an *inductive definition*, which adheres to the well-founded semantics and allow for specifying recursive concepts. Those two lines inductively define the $Reachable$ predicate, starting from the depot, and inductively adding neighbouring nodes according to the links present in $Path$. The last line then states that all nodes must belong to $Reachable$, forming a subtour elimination constraint – if a node is not reachable from the depot, it belongs to a subtour different from the one with the depot.

- The term *obj* over V representing the total travelling distance and serving as the objective function: $\Sigma_{(x,y)\in Path} Distance(x,y)$
- The structure $\bar{S}|_V$ describing type and parameter values. In $\bar{S}$, the assignment to problem parameters (in this case, $Node$ and $Distance$) represents an instance of the problem, while assignments to variable symbols (in this case, $Path$ and $Reachable$) could form a solution for that instance.

3.3 Modelling local search heuristics

All the modelling in the following sections is an extension of the modelling of problem $\mathcal{P}$ with vocabulary V, theory $T|_V$. In our framework, users have to prefix the decision variables of the problem with dec, so that the *local search* inference identifies them correctly. We use $\bar{S}|_V$ to refer to an instance of $\mathcal{P}$. To model a local search algorithm, users first need to extend the problem modelling with neighbourhood moves modelling (section 3.3.1). The IDP system provides "off-the-shelf" heuristics and metaheuristics (section 3.3.2) from which users can choose. Given a local search type and a set of neighbourhood moves, IDP will synthesize the desired algorithm by passing the neighbourhood moves to the corresponding (meta)heuristic framework.

3.3.1 Modelling neighbourhood moves

Neighbourhood moves are essential components of local search algorithms. Given a set of neighbourhood moves, many heuristics and metaheuristics can be synthesized. In this section, we present how a neighbourhood move can be modelled in FO($\cdot$) in our framework. Users can specify multiple neighbourhood moves for a problem, but each neighbourhood must have a unique name. All the modelling components of a neighbourhood must include its name as a suffix for the framework to detect the move. In this chapter, this suffix is omitted to simplify the description. Two different types of neighbourhood moves are supported: standard neighbourhood moves and Large Neighbourhood Search (LNS) moves.

Standard neighbourhood moves

For a standard set of neighbourhood moves, the following components need to be defined: (i) how to get the valid moves given a solution; (ii) how to compute a neighbouring solution given a neighbourhood move and a current solution; (iii) how to calculate the benefit of a given move (the difference between the objective values of the current solution and the neighbouring solution resulting from this move). To model those pieces of information, the following components are required in the problem modelling:

- A vocabulary $Vmove$, which consists of functions and predicates representing the move.
- A query $getValidMoves|_V$, which describes how to get valid moves from a given solution.

- A vocabulary $Vnext$, which consists of logical symbols for creating a neighbouring solution from an original solution and a move. The symbols in $Vnext$ are divided into 3 groups: (1) the symbols of V representing the current solution; (2) the symbols of $Vmove$ representing the move; and (3) a set of copies $pnext$ for each symbol $p \in V$ corresponding to a decision variable, representing the solution resulting from applying the move described in terms of $Vmove$ on the current solution described in terms of V.
- A theory $Tnext|_{Vnext}$, which consists of FO($\cdot$) definitions mapping a solution to its neighbour, given a move.
- A query $getDeltaObj|_{Vnext}$, which evaluates the benefit of a move by calculating the difference between the objective values of the current solution and its neighbour. A fast evaluation function is essential to the performance of a local search algorithm. Users can define the query to evaluate a move exactly or heuristically depending on the problem at hand.

The above components must satisfy certain formal requirements to form a valid neighbourhood move for a problem. A model of $Tnext$ contains essentially two structures: the original one expressed in the assignments of the original symbols of V, and the neighbour structure found in the assignments of the "next" symbols. In a valid neighbourhood move description, a move to a neighbouring solution preserves the hard constraints, which are specified in theory T. I.e., if M is a model of $Tnext$, and $M|_{\Sigma}$ is a model of T (hence, satisfies the hard constraints), then the restriction of $M|_{\Sigma next}$ to the "next" symbols is also a model of T and satisfies the hard constraints.

The above components are illustrated using the running example of the TSP as follows.

Example 3.3.1. In this example, we model the *2-opt* move for the TSP, where two edges are removed from the solution and replaced by two new edges (see figure 3.1). To specify a valid move, extra information regarding the order of nodes appearing along the solution path is needed. Therefore, the problem specification is extended with predicate $Before$ to express this information.

1. $\forall x : Before(x, x).$
2. $\forall x, y : Before(x, y) \qquad \leftarrow Path(x, y) \wedge y \neq Depot.$
3. $\forall x, y : Before(x, y) \qquad \leftarrow \exists z : Path(x, z) \wedge Before(z, y) \wedge z \neq Depot.$

$Vmove$, the vocabulary for representing the move, consists of 4 constants $S1, E1, S2$ and $E2$ representing the four nodes involved in the *2-opt* move. $(S1, E1)$ and $(S2, E2)$ are two edges in the solution in this specific order and there may be a number of nodes between $E1$ and $S2$. Query $getValidMoves$ gets all valid moves from a given solution,

which are two edges appearing in the respecting order on the solution path starting from the depot. $Vnext$, besides the symbols from V representing the original solution and $Vmove$ representing a move, consists of the predicate $next_Path(Node, Node)$, which represents a neighbouring solution.

Theory $Tnext$ consists of an inductive definition describing how to create a neighbouring solution given a solution and a move. In particular, the new solution is expressed by the predicate $next_Path$. Its first segment is the current segment from the depot up to $S1$ (expressed by rule $*1$), followed by the move from $S1$ to $S2$, (expressed by rule $*2$), followed by the path from $S2$ to $E1$ obtained by inverting the current path from $E1$ to $S2$ (expressed by rule $*3$), followed by a new edge from $E1$ to $E2$ (expressed by rule $*4$), and final segment from $E2$ back to the depot (rule $*5$).

1. $next_Path(x, y) \leftarrow Path(x, y) \wedge Before(y, S1) \wedge y \neq Depot.$
2. $next_Path(S1, S2).$
3. $next_Path(y, x) \leftarrow Path(x, y) \wedge Before(E1, x) \wedge Before(y, S2) \wedge y \neq Depot.$
4. $next_Path(E1, E2).$
5. $next_Path(x, y) \leftarrow Path(x, y) \wedge Before(E2, x) \wedge E2 \neq Depot.$

The last piece of the description, query $getDeltaObj$, evaluates a move by calculating the delta between the total travelling time of the current solution and its neighbour: $\Delta = d_{S1S2} + d_{E1E2} - (d_{S1E1} + d_{S2E2})$.

We hereby prove that the above description satisfies the formal requirement of a valid neighbourhood move for the TSP. Given a partial interpretation of $Vnext$ M, $M|_V$, or to be more specific, the interpretation of $path$ in $Vnext$ encodes the original path of the original solution, while $M|_{Vmove}$ encodes the considered move. If the original solution is a valid TSP solution, $M|_V$ then satisfies all hard constraints specified in the original theory T, e.g. the assignment constraints and subtour constraints. Theory $Tnext$ specifies how the interpretation of $path_next$ is created from the interpretation of $path$ and the constants given by the move. It states that the new solution consists of the segment from the depot to $S1$, the edge $(S1, S2)$, the reversed segment from $E1$ to $S2$, the edge $(E1, E2)$, and finally the segment from $E2$ to the depot (an illustration can be found in figure 3.1). Given that the move and the original path is valid, then the new tour formed by $next_Path$ does not contain any subtour and also the assignment constraints are satisfied. All the hard constraints defined in T are satisfied, making it a valid solution for the TSP. The neighbourhood description hence meets the formal requirement of a valid neighbourhood move.

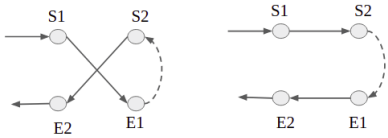


Figure 3.1: The 2-opt move.

LNS moves

Large Neighbourhood Search (LNS) (Shaw, 1998) is a technique that allows for exploring a large neighbourhood of the current solution. LNS consists of a *destroy* and a *recreate* phase, which are alternated to gradually improve a given initial solution. In the destroy phase, a (usually large) part of the solution is destroyed. In the recreate phase, the partial solution is then repaired using different techniques. Originating from the Constraint Programming community, LNS conventionally uses a black-box solver in the recreate phase to repair the solution. Some logic and constraint programming systems support automatic LNS (Perron et al., 2004) (Pisinger and Ropke, 2010), where the parts of a solution to be destroyed are selected randomly or heuristically by the solvers. However, allowing users to specify which parts of the solution to be destroyed is desirable, since destroying the bad part of the solution is more likely to lead to better solutions. Therefore, in our framework, we support LNS moves by allowing users to model the destroy phase, while the recreate phase is handled by the solver.

To get an intuition on how the destroy phase should be modelled, let us consider an example of the nurse scheduling problem (NSP) where nurses with different skills need to be assigned to shifts, subject to some hard and soft constraints. An example of a meaningful LNS move for the NSP is to destroy the schedules of two nurses in the destroy phase and reschedule them in the recreate phase. The two nurses are usually selected randomly from a set of nurses whose preferences are violated. To model this move, users should be allowed to specify how the random factors should be chosen (e.g. nurses whose preferences are violated). Given selected values of the random factors (e.g. nurses), users can specify that the schedules of those nurses will be destroyed in the first phase of the LNS.

With that intuition in mind, users can model an LNS move in our framework by specifying: (i) which variables play the role of random factors in the heuristics; (ii) how to select domain values of those random variables; and (iii) which parts of the solution should be destroyed given the selected domain values of the random variables. The first piece of information (i) is encoded in a vocabulary $Vmove$. (ii) is given through a query $getRandomVars$. By solving the query, a set of possible options for the domain elements of the random variables is obtained, from which the solver selects a random set of values. Given the chosen domain elements (values) of the random variables,

users can specify parts of the solution to be destroyed (iii) through query $getMoves$. An example of an LNS move modelling of the running example of the TSP is presented below.

Example 3.3.2. Each solution of the TSP is a Hamiltonian path that visits all vertices of the graph. Let us say users want to destroy a part of this path, from node S to node E, given that S appears before E in the path starting from the depot. Vocabulary $Vmove$ then consists of two constants S and E. S and E are random factors, which are chosen at each iteration of the algorithm. Query $getRandomVars$ specifies valid values of S and E:

$$\{s, e \mid Before(s, e) \wedge s \neq e \wedge e \neq Depot\}$$

Given the chosen values of S and E, the part to be removed from the solution in the destroy phase of the LNS is the path from S to E, which is encoded in query $getMove$:

$$\{x, y \mid Before(S, x) \wedge Before(y, E)\}$$

Besides a user-defined LNS move, the framework also supports automatic LNS moves, where parts of the solution to be destroyed are chosen randomly.

3.3.2 Local search algorithms in IDP

In our IDP-based framework, we implemente some “off-the-shelf" heuristic and metaheuristic algorithms. Our framework takes neighbourhood modellings as input and synthesizes them accordingly with the chosen (meta)heuristics. The following local search algorithms are implemented:

- *First improvement search* and *best improvement search*: two simple heuristics which iterate through all valid moves and proceed with a move once the first (or the best) improvement of the objective value is found.
- *Tabu search* (Glover and Laguna, 1998): a more advanced metaheuristic which keeps a list of forbidden moves to avoid reconsidering recently visited solutions.
- *Large Neighbourhood Search* (LNS) (Shaw, 1998): a metaheuristic that explores large neighbourhoods by destroying and repairing a solution iteratively. LNS takes only LNS moves as inputs. Automatic LNS moves are also implemented in our framework.

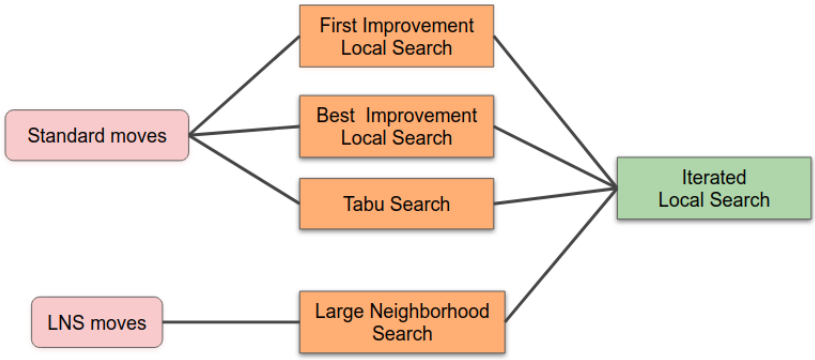


Figure 3.2: The built-in local search algorithms in IDP.

- *Iterated Local Search* (ILS) (Lourenço et al., 2003): a metaheuristic which allows for the composition of (meta)heuristics. The embedded (meta)heuristics are executed iteratively, each takes the best solution returned by the preceding (meta)heuristics as the initial point and stops when a termination condition is met. ILS is a powerful metaheuristic which allows for combining many local search algorithms, corresponding to different neighbourhood moves.

All supported local search algorithms adopt two stopping criteria, namely maximum running time and a maximum number of iterations without improvement. Figure 3.2 summarizes the supported neighbourhood moves, local search algorithms and their relationships.

3.4 Local search framework

In this section, we provide insight into how IDP is extended to simulate a local search algorithm from its modelling. We first present how a simple first improvement search (FI) with a single standard neighbourhood is simulated in IDP. The simulation of other local search algorithms with standard neighbourhood moves is straightforward given the description of FI. The simulation of the LNS is also straightforward from the description of LNS moves in section 3.3.1.

Local search with standard neighbourhood moves Let us first recall the components of the modelling. A problem's modelling consists of a vocabulary V, a theory T, a query $getObjVal$ and a term obj. Each neighbourhood move modelling

consists of a vocabulary $Vmove$, a query $getValidMoves$, a theory $Tnext$, and finally a query $getDeltaObj$. The simulation of the FI local search is presented in algorithm 1.

Algorithm 1: FI local search in IDP with a single standard neighbourhood move

input : instance α
output best solution s^*
:

```
1  s_0 ← get an initial solution by applying modelexpand(T, α);
2  s ← s_0;
3  s* ← s_0;
4  repeat
5      Ω ← get a set of valid moves by calculating query getValidMoves on s;
6      repeat
7          ω ← select a move from Ω;
8          δ ← evaluate ω by solving query getDeltaObj;
9          if ω is accepted then
10             s' ← create new solution from ω and s;
11             s ← s';
12             update best solution s*;
13             break;
14         end
15     until no valid move left to be considered;
16 until no improvement found or timeout;
```

In line 1, given an instance α, IDP applies $modelexpansion$ inference to obtain the first feasible solution, which will serve as the initial point s_0 of the local search heuristic. In line 5, a set of valid neighbourhood moves of the current solution s (denoted as Ω) is achieved by solving the query $getValidMoves$, using IDP's $querysolving$ inference. The way Ω is explored is determined by the local search type. In the case of FI local search, an unexplored move ω is selected from Ω. ω is evaluated by solving the query $getDeltaObj$ on s_ω, where s_ω is a joined structure between the current solution s and the move ω (line 8). Given the obtained δ, the move is then evaluated based on the acceptance criteria of the local search algorithm. If ω is accepted, the neighbouring solution is created by applying the procedure $createneighbour$. The best solution s^* and the current solution s are then updated accordingly. The loop then continues with the new starting point s and terminates when a stopping criterion is met. A visualization of the local search procedure can be found in figure 3.3.

Creating a neighbouring solution Given a neighbourhood move ω and a current solution $s|_V$, a neighbouring solution s' is created as follows. Recall that to evaluate a move ω, a structure s_ω is created by merging ω and s. s_ω is a partial interpretation of $Vnext$. In $Vnext$, the prefix $next_dec$ indicates the neighbour

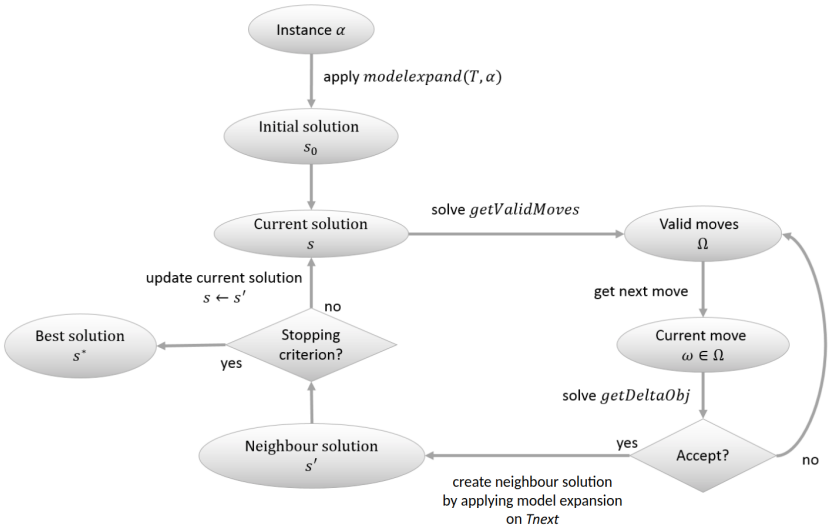


Figure 3.3: The local search framework in IDP.

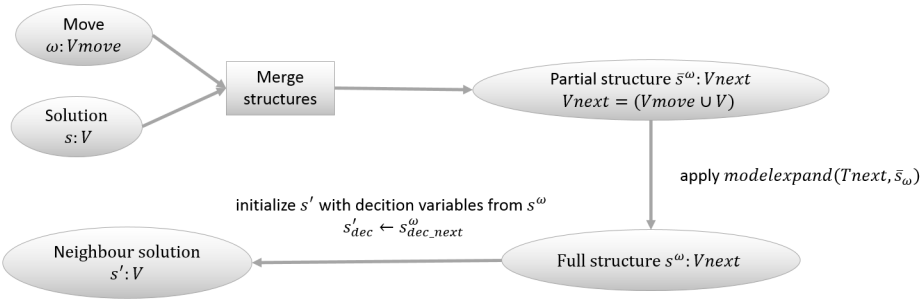


Figure 3.4: Creating a neighbouring solution in IDP.

decision variables, whose values define the corresponding neighbouring solution. By applying $modelexpansion$ on s_ω over theory $Tnext$, an interpretation of those variables is achieved. To create the neighbouring solution s' corresponding to those values, the interpretation of the neighbour decision variables from s_ω is extracted and is used to initialize the values of the decision variables of s'. The procedure is illustrated in figure 3.4.

To explain the procedure in more detail, we once again use the running example of the TSP.

Example 3.4.1 (TSP)**.** Given an instance α with 5 nodes A, B, C, D and E and a solution $s|_V$ with the solution path $\{A, B, C, D, E\}$, the decision variables of s hold the following values: $s^{dec_Path} = \{A, B; B, C; C, D; D, E\}$.

A 2-opt move ω hold the values: $s^{S1,E1,S2,D2} = \{B\}; \{C\}; \{D\}; \{E\}$ which means the move will replace two edges (B, C) and (D, E) by (B, D) and (C, E), resulting in the new solution path $\{A, B, D, C, E\}$. By merging s and ω, the resulting structure $s_\omega|_{Vnext}$ will be a partial interpretation of $Vnext$ with domain values from both structures. By applying model expansion on s_ω over $Tnext$, the next decision variables of s_ω will take the values: $s_\omega^{next_dec_Path} = \{A, B; B, D; D, C; C, E\}$. Those values make up the decision variables of the neighbouring solution. To create a neighbouring solution s', we clone a new structure s' from the input instance α and initialize its decision variables by the values extracted from $s_\omega^{next_dec_Path}$:

$$s'^{dec_Path} \leftarrow s_\omega^{next_dec_Path}$$

s' is then the neighbouring solution corresponding to the solution path $\{A, B, D, C, E\}$.

In the above simulation, the potential bottleneck for the performance of the algorithm comes from the use of *query solving* and *model expansion* of IDP. On some problems, if the query to get valid moves or to evaluate a move is too complicated, it might take a long time to solve the query which leads to less iterations performed by the local search algorithm within the time limit. On other problems, *model expansion*, for which each call requires a grounding phase, which is very expensive, might lead to limited performance on large instances. Therefore, declarative local search in IDP can be improved if those two inferences are improved (e.g., by reusing existing groundings for consecutive model expansions).

3.5 Experimental results

In this section, we benchmark the framework using the TSP running example. The motivation for this choice is twofold. First, the TSP is a classical optimization problem which has been investigated intensively in the literature of operations research. However, the TSP (and its variants) is still a challenge for logic-based systems such as IDP or ASP solvers (Devriendt, 2017). Through our empirical results, we prove that our local search backend is a good complement for the existing mechanisms of logic programming on such problems. Second, with a rich set of efficient and easy-to-model neighbourhood moves (Mak and Morton, 1993), the TSP is very suitable to analyse and demonstrate the framework.

The dataset used in this experiment is taken from the TSPLIB (Reinelt, 1991). 80 instances, with size ranging from 14 to 443 nodes, are selected as the testbed. The experiments are conducted on an Intel Core i7-4600 2.90GHz * 4 machine with 8GiB of RAM, running Ubuntu 16.04 64-bit. The time budget for each run for all test algorithms is 600 seconds. The solver, modelling along with test instances and results can be found in https://github.com/tusanpham/DeclarativeLocalSearch.

The neighbourhoods modelled in this experiment are listed below:

- *2-opt* move.
- *oropt1* (or relocate move) and *oropt2*, where a node or two consecutive nodes are relocated to a different position in the tour.
- *lns1*, an LNS move where a random segment of the solution is removed from the solution.
- *lns2*, an LNS move where a set of closest neighbours of a random node are removed from the solution.

From those moves, 15 algorithms are constructed:

- Three local search configurations (based on *first improvement*, *best improvement* and *tabu search*) for each standard neighbourhood move (*2opt*, *oropt1* and *oropt2*).
- Two *LNS* configurations corresponding to two LNS moves *lns1* and *lns2*.
- Four *ILS* configurations with different local search combinations.
  - *ILS1*: combines *FI* using *oropt1* (denoted as *FI(oropt1)*), *FI(2opt)*, and *FI(oropt2)* in this order.
  - *ILS2*: combines *FI(2opt)*, *FI(oropt1)*, and *FI(oropt2)* in this order.
  - *ILS3*: combines *FI(2opt)*, *FI(oropt2)*, and *FI(oropt1)* in this order.
  - *ILS4*: combines *FI(oropt1)*, *best improvement(2opt)* and *tabu search(2opt)* in this order.

The 15 algorithms are compared to IDP's *minimization* and clingo (Gebser et al., 2014). The results are summarized in figure 3.5. The y-axis shows the deviation from the obtained solutions to the optimal solutions, the lower the better. It can be seen clearly from the results that most tested local search algorithms perform better than IDP and clingo. Regarding simple local search algorithms using a single neighbourhood move, *first improvement* using *2opt* move and *tabu search* using *2opt* move seem to perform

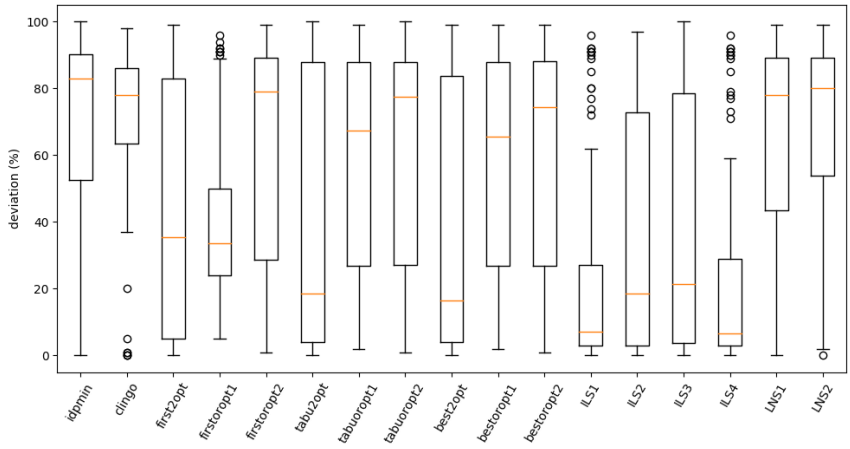


Figure 3.5: Results for the TSP.

the best. Among all test algorithms, *ILS1* and *ILS4* clearly stand out from the rest with the average deviation less than 10%, which is 7 or 8 times better than IDP and clingo. This result is firstly a demonstration of the potential use of the fast prototyping capacity of the framework. With very simple modellings, different local search heuristics can be composed and analysed. Secondly, the big gap between *ILS1* and *ILS4* and the two black-box logic solvers supports our claim that declarative local search could be a good complement to such logic solvers.

3.6 Discussion

In this section, we discuss the limitation of the framework and propose potential solutions to improve it accordingly. Even though the synthesized local search algorithms outperform the two logic solvers IDP and clingo, there is still an enormous gap between them and the ad-hoc implementations of the same algorithms in C++. There are two reasons for this. First, the trade-off between expressiveness and efficiency is well-known in Knowledge Representation. FO(·), the input language of the framework, is a formal expressive language that aims for general problems rather than combinatorial optimization. The generic data structures of IDP are not optimized for optimization problems. To be more specific, they do not support incremental computation of objective values, which is essential for local search algorithms. Even though currently users can "smartly" define a heuristic evaluation function through the *getDeltaObj* query, such incremental computation is still necessary to close the gap between a synthesized algorithm and an ad-hoc implementation. Second, the process of

synthesizing algorithms employs IDP's *model expansion* which requires grounding the theory T_next at each iteration. The expensive grounding step is a bottleneck that limits the performance of the framework on large or even moderate-sized instances.

From the above analyses, an obvious solution to overcome the performance issue is to avoid the grounding step at each iteration. Recall that local search algorithms apply local changes to a solution at each iteration, so usually the current solution is almost identical with its neighbour, with only few differences. This brings up an idea of applying *model revision* to compute a neighbouring solution given a current one, instead of applying *model expansion*. Model revision is a topic of Logic Programming (Garcia et al., 2019, Gouveia et al., 2018) which investigates methods to modify a model taking into account new changes with the least computational efforts. Model revision has been studied and considered to be implemented in IDP before (Jansen, 2014, Wittocx et al., 2009). However, at the moment IDP does not support model revision yet. Once being introduced to the system, it would make a perfect solution to avoid the expensive grounding step at each local search iteration. Another potential solution is to build an internal data structure for the local search procedure which supports incremental evaluation. Similar ideas can be found in CBLS systems such as Objective-CP (Van Hentenryck and Michel, 2013) or Oscar.cbls (De Landtsheer and Ponsard, 2013).

To conclude the discussion session, we would like to highlight that even with the limitation in the performance of IDP right now, the results achieved by the synthesized algorithms still far bypass IDP and clingo, one of the most efficient ASP solvers currently, on the test problem. This is a strong indication of the great potential of declarative local search.

3.7 Conclusion and future work

In this chapter, we introduce a local search framework that executes local search heuristics from their formal, declarative descriptions in predicate logic. The framework introduces local search, a successful mechanism from the operations research community, to a logic solver. It serves as a good alternative for existing solving mechanisms of logic solvers, demonstrated by experimental results on the TSP. It also serves as a good platform for fast prototyping local search heuristics and is a step towards purely declarative local search. Future work includes extending the framework to allow for more flexibility in metaheuristics modelling – e.g. stopping criteria and adaptive selection of neighbourhood moves. The combination between user-defined neighbourhoods and automatically generated neighbourhoods (Devriendt et al., 2015) is also interesting. Due to the bottleneck of the underlying ground-and-solve system, our

framework is not yet able to solve very large scale problems. However, our framework can serve as a fast prototype platform to experiment and analyse local search heuristics.

Part II

Hybrid algorithms for specific case studies

Chapter 4

The Intermittent Travelling Salesman Problem

Goal of the chapter In this chapter we introduce a new variant of the Travelling Salesman Problem, namely the Intermittent Travelling Salesman Problem, which is inspired by real world drilling/texturing applications. In this problem, each vertex can be visited more than once and there is a temperature constraint enforcing a time lapse between two consecutive visits. A branch-and-bound approach is proposed to solve small instances to optimality. We furthermore develop four Variable Neighbourhood Search metaheuristics which produce high quality solutions for large instances. An instance library is built and made publicly available.

The content of this chapter is based on the publication:

- Tu-San Pham, Pieter Leyman, and Patrick De Causmaecker. The intermittent travelling salesman problem. *International Transactions in Operational Research*, pages 525–548, 2018b.

4.1 Introduction

In this chapter, we study a new variant of the Travelling Salesman Problem (TSP). The problem is inspired by industrial drilling or texturing applications where the temperature of the work-piece is taken into account. The processing causes the work-

piece to locally heat up until it eventually melts the work-piece down. Therefore, the processing must be divided over time intervals, with time lapses for cooling down of the work-piece in between. This gives rise to an optimisation problem of finding the best schedule to process the whole work-piece in the shortest time while respecting temperature constraints. In this chapter, we study a simplified version of the problem where we assume linearity with identical and constant cooling and heating speeds. We model the problem as a variant of the TSP in which some vertices are visited multiple times and there is a time lapse between two consecutive visits of a vertex, which is caused by the maximum temperature constraint, hence the name Intermittent Travelling Salesman Problem (ITSP).

The TSP is a well-studied problem in the field of combinatorial optimization with several variants. Extended reviews of the TSP can be found in e.g. (Laporte, 1992) and (Johnson and McGeoch, 1997). To the best of our knowledge, the ITSP is the first variant with intermittent constraints. The ITSP also bears similarities to the Vehicle Routing Problem family. The following problems share similarities with the ITSP. We point out the important differences.

- The TSP with multiple visits (Gutin and Punnen, 2006): In this problem, a vertex must be visited a number of times. However, unlike the ITSP, each visit is not associated with a processing time and there is no time constraint between visits.
- The TSP with Time Windows (Dumas et al., 1995) (TSPTW): The TSPTW has time constraints associated with each vertex in the form of a time window. In contrast to the ITSP, the TSP with time windows does not consider multiple visits and the time constraints apply only for individual vertices.
- The Inventory Routing Problem (Campbell et al., 1998) (IRP): Each vertex (retailer) has a periodical (e.g. daily) requirement and an inventory capacity that should not be exceeded. The problem consists of finding the best shipping policy to deliver a product from a common supplier to several retailers, subject to the vehicle capacity constraints, product requirements and inventory capacities of the retailers. The inventory constraints are similar to the temperature constraints, be it that in the IRP, vehicle travelling routes are within one discrete period (e.g. a day). Therefore, route scheduling does not affect retailer inventory levels, in contrast with the ITSP where travelling time directly affects vertex temperature.

In this chapter, we provide a detailed description of the ITSP and analyse its properties. We study a decomposition model, a branch-and-bound approach and a metaheuristic. A decomposition model is proposed including a Mixed Integer Programming (MIP) and a Linear Programming formulation for the sub-problems. These are integrated in a branch-and-bound approach shown to solve small instances to optimality. We then develop a metaheuristic approach for the larger instances. Two constructive heuristics

are proposed. Those two heuristics and two perturbation strategies are integrated in a Variable Neighborhood Search (VNS) framework resulting in four VNS metaheuristics. We analyse problem difficulty and create a problem library as a foundation for further research.

The structure of the chapter is as follows. Section 4.2 describes the ITSP in detail and analyses its difficulty. Section 4.3 introduces the building blocks of the branch-and-bound approach, namely a lemma, two sub-problems and two corresponding mathematical models. Section 4.4 is devoted to the Variable Neighborhood Search approach. Experimental results are given in section 4.5 and finally a conclusion and future work follow in section 4.6.

4.2 Problem description

We consider the problem of visiting a number of vertices $v \in V$ with a given distance matrix $c_{vu} \geq 0$ defined for every pair of vertices $(v, u) \in V \times V$ with $c_{vv} = 0$. We assume the triangle inequality holds. Each vertex $v \in V$ needs to be processed for a certain time d_v. Let t, t_s and t_e be the current time, the beginning and the end of the current contiguous period of processing respectively. While being processed, the temperature of a vertex rises linearly, $T_v(t) = T_v(t_s) + (t - t_s)$. While not being processed, the temperature cools linearly, $T_v(t) = max(T_v(t_e) - t, 0)$ with a minimum of 0, below which no cooling occurs. The maximum temperature of a vertex is B, beyond which no processing is allowed. We assume that at the beginning of the processing, the temperature of each vertex is 0. The ITSP asks for the processing schedule of shortest total completion time.

We introduce some terminologies used throughout the chapter:

- *A processing-step* is a processing interval without interruption at a vertex.
- *A visit* consists of one or more processing-steps at one vertex without any visits to other vertices in between.

The term 'seconds' is used to indicate time units. Due to the maximum temperature constraint, a *processing-step* can never exceed B seconds. For example, a vertex v needs to be processed for 20 seconds while the maximum temperature B allowed is 10 degrees. We let the increase in temperature during processing be one degree per second. Given this linear increase and decrease, v must be processed in at least two blocks of 10 seconds with a time lapse of at least 10 seconds in between. Hence, we can say v is processed using at least two *processing-steps*. If between those two *processing-steps* there is only waiting time and no processing-steps at other vertices, we say node v is processed by one *visit*. Otherwise, we say v is processed by two *visits*.

For example, if the sequence of processing for an instance consisting of 2 vertices is $A - A - B$ with a waiting time between two processing intervals of A, we say A is processed in one visit and this visit includes two processing-steps.

We hereby discuss the number of processing-steps in optimal solutions. The duration of a processing-step must be less than or equal to B time units where B is the maximum temperature allowed. This results in the minimum number of processing-steps required for a vertex v (denoted by ϕ_v), with processing time d_v, given by:

$$\phi_v = \lceil d_v / B \rceil$$

In some instances where the maximum temperature is significantly smaller than the average travelling time between vertices, we may have to divide the processing of v into more than ϕ_v processing-steps.

Such an example can be seen in figure 4.1. In this instance, vertex A has a processing time $d_A = 38$ seconds while four other vertices all have a processing time of 1 second. The maximum temperature B is 10 degrees. There are two solutions considered: S_1 where A is processed in 5 processing-steps (denoted by $A_i, i = \{1, .., 5\}$, with the corresponding processing times denoted by p_{A_i}); and S_2 where A is processed in 4 processing-steps ($A_i, i = \{1, .., 4\}$), i.e. the minimum number of processing-steps required, with positive waiting times at the third and the fourth processing-steps ($w_{A_3}, w_{A_4} > 0$). In the figure, $[v]$ denotes the tour time to move from vertex A to vertex v, process v and then come back to A. Despite the larger number of processing-steps, S_1 has a better objective value (66) than S_2 (69) due to no waiting time between processing-steps.

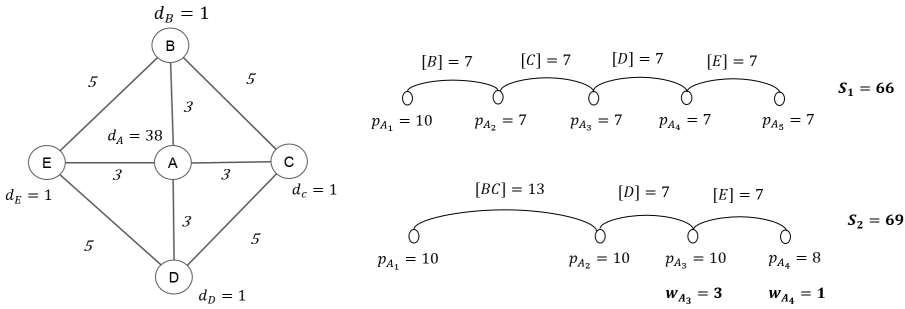


Figure 4.1: Sample instance illustrating a larger number of processing-steps may yield a better solution.

4.3 Branch-and-bound approach

4.3.1 Mathematical models and lemmas

In this section, the building blocks of the branch-and-bound approach, namely a lemma, two subproblems and two mathematical models to solve the subproblems, are introduced.

Subproblem 1: a fixed number of processing-steps

Given an instance and a fixed number m of processing-steps, the optimal solution can be obtained using the MIP model below.

Consider a set of processing-steps $Q = \{0, .., m-1\}$, each processing-step $j \in Q$ is associated with a vertex $v \in V$. d_v is the required processing time of vertex v and c_{vu} is the cost (in time units) to travel from vertex v to vertex u.

The model consists of the following variables:

- y_{jv} (binary): $y_{jv} = 1$ iff vertex v is processed during processing-step j.
- x_{jvu} (binary): $x_{jvu} = 1$ iff vertex v is processed during processing-step j and vertex u is processed during processing-step $j + 1$.
- s_{jv} (continuous): the processing time of processing-step $j \in Q$ at vertex $v \in V$.
- t_j (continuous): the time at the beginning of processing-step $j \in Q$.
- q_{jv} (continuous): the temperature of vertex v at the beginning of processing-step $j \in Q$.

The objective is to minimize the total completion time, which is the sum of the starting time of the final processing-step and its processing time. The model is as follows:

$$\text{minimize } t_{m-1} + \sum_{v \in V} s_{m-1,v} \tag{4.1}$$

subject to

$$y_{jv} = \sum_{u \in V} x_{jvu} \qquad \forall j \in Q, v \in V \tag{4.2}$$

$$y_{jv} = \sum_{u \in V} x_{j-1,uv} \qquad \forall j \in Q | j > 0, v \in V \tag{4.3}$$

$$\sum_{v \in V} y_{jv} \leq 1 \qquad \forall j \in Q \tag{4.4}$$

$$y_{0,0} = y_{m-1,0} = 1 \tag{4.5}$$

$$t_{j+1} \geq t_j + \sum_{v \in V} s_{jv} + \sum_{v,u \in V} c_{vu} x_{jvu} \qquad \forall j \in Q | j \neq m-1 \tag{4.6}$$

$$\sum_{j \in Q} s_{jv} = d_v \qquad \forall v \in V \tag{4.7}$$

$$q_{j+1,v} \geq q_{jv} + 2s_{jv} - (t_{j+1} - t_j) \qquad \forall j \in Q | j \neq m, \forall v \in V \tag{4.8}$$

$$q_{jv} + s_{jv} \leq B \qquad \forall j \in Q, v \in V \tag{4.9}$$

$$s_{jv} \leq B y_{jv} \qquad \forall j \in Q, v \in V \tag{4.10}$$

$$x_{jvu} \in \{0, 1\} \qquad \forall j \in Q, v, u \in S \tag{4.11}$$

$$y_{jv} \in \{0, 1\} \qquad \forall j \in Q, v \in V \tag{4.12}$$

$$0 \leq s_{jv} \leq B \qquad \forall j \in Q, v \in V \tag{4.13}$$

$$0 \leq t_j \qquad \forall j \in Q \tag{4.14}$$

$$0 \leq q_{jv} \leq B \qquad \forall j \in Q, v \in V \tag{4.15}$$

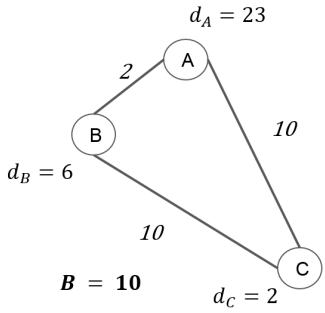
where (4.2) and (4.3) are linking constraints between x and y. (4.4) are flow constraints while (4.5) imposes the starting and ending depot (vertex 0). Constraints (4.6) define arrival times. Constraints (4.7) ensure total processing times of vertices. Constraints (4.8) define arrival temperature of processing-steps. This equation comes from the linear increase and decrease of temperature, which results in the temperature at the beginning of the processing-step $j + 1$ being defined as $q_{j+1,v} = max(q_{jv} + s_{jv} - (t_{j+1} - (t_j + s_{jv})), 0)$. (4.9) are temperature constraints and (4.10) are linking constraints between s and y. The rest are range constraints.

Subproblem 2: a fixed sequence of visits

Given a fixed sequence of visits, the optimal completion time can be obtained in polynomial time using a Linear Programming (LP) model. Note that, unlike the MIP

model in subproblem 1, this model only needs to consider *visits* eventually aggregating a number of *processing-steps*.

An example of a fixed sequence of visits can be found in figure 4.2. The sequence has 4 visits, each of which is associated with a vertex, a processing time and a waiting time. Visit 0, 1 and 3 include a single processing-step whose processing time is smaller than or equal to the maximum temperature B (10 time units) while visit 2 includes 2 processing-steps with total processing time $p_2 = 13 > B$. Since its total processing time is larger than the maximum temperature, a waiting time of $w_2 = B - p_2 = 3$ time units is required.



Visit i	0	1	2	3
Vertex v	A	B	A	C
p_i	10	6	13	2
w_i	0	0	3	0

Figure 4.2: A fixed sequence of visits.

Given is a fixed sequence Σ of n_Σ visits in which every vertex $v \in V$ is visited at least once. Let w_i, p_i and q_i represent the waiting time, the total processing time and the temperature at arrival of visit i ($i \in \{0, .., n_\Sigma - 1\}$). Given a visit i associated with vertex $v \in V$, pre_i denotes the previous visit associated with the same vertex (in the example in figure 4.2, $pre_2 = 0$). Π_v is the set of visits associated with vertex $v \in V$. An optimal visiting schedule of Σ, i.e. processing times and waiting times of its visits, can be calculated using the Linear Programming model as follows.

$$\text{minimize} \sum_{i=0}^{n_\Sigma - 1} w_i \tag{4.16}$$

subject to

$$w_i \geq q_i + p_i - B \qquad i \in \{0, ..., n_\Sigma - 1\} \tag{4.17}$$

$$\sum_{i \in \Pi_v} p_i = d_v \qquad v \in V \tag{4.18}$$

$$q_i \geq (q_{pre_i} + (p_{pre_i} - w_{pre_i}) - c_{pre_i, pre_i+1} - \sum_{k=pre_i+1}^{i-1} (c_{v_k v_{k+1}} + p_k + w_k)) \qquad i \in \{0, .., n_\Sigma - 1\} \tag{4.19}$$

$$w_i, p_i, q_i \geq 0 \qquad i \in \{0, ..., n_\Sigma - 1\} \tag{4.20}$$

The objective function (4.16) minimizes the total waiting time. Constraints (4.17) come from the statement of vertex temperature, namely that temperature upon arrival (q_i) + processing time (p_i) - waiting time (w_i, corresponds with cooling down of vertex) should never be larger than the maximum temperature (B). Constraints (4.18) make sure each vertex is processed exactly its required amount of time. Constraints (4.19) limit the temperature at the start of the i^{th} visit from below by considering the arrival temperature (q_{pre_i}), total processing time and waiting time ($p_{pre_i} - w_{pre_i}$) of its previous visit and the tour time (travelling times plus processing times and waiting times) of all visits between pre_i and i (the remaining part of the equation). The program can be solved in polynomial time in the number of variables which is proportional to n_Σ.

Adding a visit to a sequence results in an increase in its total travelling time. For any three vertices $v, x, y \in V$, let t_{vxy} represent the travelling time increased when a visit of v is added between two consecutive visits of x and y in a sequence:

$$t_{vxy} = c_{xv} + c_{vy} - c_{xy}$$

We have the following:

Lemma 1. *If, for a given sequence Σ, the optimal total waiting time $W_\Sigma = \sum_{i=0}^{n_\Sigma - 1} w_i$ is less than or equal to $min_{v,x,y \in V}\{t_{vxy}\}$, no extension of Σ generated by adding an extra visit can produce a shorter completion time.*

Proof. Adding an extra visit increases the total travelling time by at least $min_{v,x,y \in V}\{t_{vxy}\}$ and reduces the waiting time by at most W_Σ.

Given the condition $W_\Sigma \leq min_{v,x,y \in V}\{t_{vxy}\}$, the total processing time is increased. □

4.3.2 Branch-and-bound approach

Based on the above subproblems and lemma, a branch-and-bound approach is proposed to solve the ITSP. We first present the data structure used in the algorithm.

Each node of the search tree is a sequence of visits. Given a sequence, its schedule, i.e. processing time and waiting time at each visit, is obtained using the LP model from subproblem 2. Since the total processing time is the same for all nodes in the search tree, we omit it to simplify the model. A node ψ in the search tree is a direct child of a node χ if ψ is created by inserting one more visit into the sequence of χ. Since the triangle inequality holds, given a node χ with the total travelling time T_χ, the total travelling time of each of its descendants is larger than or equal to T_χ. Hence, we can say T_χ is a valid lower bound for χ and all its descendants.

$$LB_\chi = T_\chi$$

A node χ can be safely pruned when $LB_\chi \geq UB$ where UB is the global upper bound of the search tree, i.e. $T_{\chi^*} + W_{\chi^*}$ where χ^* is the best solution so far. A node is also pruned when the condition in lemma 1 is satisfied.

At the beginning of the search, the MIP model in subproblem 1 is solved with the minimum number of processing-steps Φ, which is calculated by adding up the minimum number of processing-steps of all vertices.

$$\Phi = \sum_{v \in V} \phi_v$$

The minimum number of processing-steps Φ ensures a feasible solution is always found. The solution given by subproblem 1 at this step provides an incumbent solution, and hence the initial upper bound (UB) of the search tree.

The search is conducted in a best first search strategy using a priority queue whose elements are always sorted ascendingly by the travelling times of their sequences. The queue is initialized by all TSP sequences (i.e. all Hamiltonian tours of the input graph) whose travelling times are smaller than UB.

At each iteration, the sequence at the head of the queue is considered. If the pruning condition in lemma 1 is satisfied, the node is pruned. Otherwise, the branching is

conducted by generating and inserting all children of the sequence considered into the priority queue so that the ascending order of travelling times is still respected. In this manner, by the time the search terminates, the travelling time of the last sequence processed is the valid lower bound for the rest of the queue.

Every time the number of processing-steps of the considered sequence is larger than the processing-steps of the best solution, the MIP model in sub-problem 1 is solved with the new number of processing-steps to obtain a new ITSP solution. If the new solution is better than the best solution, the best solution and UB are updated accordingly. The search terminates when the gap between the UB and LB is closed or all nodes have been searched.

Since during the branching, different paths can lead to the same sequence, explored sequences are kept in a *visited archive* which is updated incrementally during the search. The algorithm is described in algorithm 2.

In line 1, the MIP model in subproblem 1 is solved with the minimum number of processing-steps Φ. The resulting solution is the first incumbent solution which provides the initial upper bound UB. In line 3, lower bound LB is obtained by solving the TSP on the original instance using Concorde (Applegate et al., 2006). If the gap is not closed after the initial phase, the priority queue Q is initialized with TSP sequences whose travelling times are smaller than UB (line 7). The *visited archive* is initialized in line 8 to keep track of the visited sequences. The search is repeated until Q is emptied or the gap is closed. At each search step, the sequence at the head of the queue is selected and removed from the queue (line 10). LB is then updated accordingly (line 11 to 13). The LP model in sub-problem 2 is solved to obtain the corresponding ITSP solution (line 14). If the number of processing-steps of the obtained solution is larger than the number of processing-steps of the best solution, the MIP model in sub-problem 1 is solved with the new number of processing-steps (line 16). The best solution and UB are then updated accordingly (line 17-19). If lemma 1 is not satisfied, the current sequence is branched, i.e. children sequences are generated and inserted into the queue (line 21 to 29). Otherwise, the branching stops and the current sequence is pruned.

4.4 Variable Neighborhood Search

Variable Neighborhood Search (VNS) (Mladenović and Hansen, 1997) is a metaheuristic framework based on the systematic change of neighborhoods during the search in order to escape from the valleys containing local optima. The basic VNS uses a set of neighborhood structures denoted by $N_k, k = \{1, .., k_{max}\}$. At each iteration, a random solution from the k^{th} neighborhood is generated. A local search procedure, which takes this random solution as the starting point, is then called to obtain a local optimum. The process repeats until some stopping criteria are met. The basic VNS framework is

Algorithm 2: Branch-and-bound search.

input : instance ins

$bestsol \leftarrow$ solve subproblem 1 with the minimum number of processing-steps Φ;
$UB \leftarrow$ objective value of $bestsol$;
$LB \leftarrow$ TSP travelling time given by Concorde;
if $LB = UB$ **then**
    optimal solution found, exit;
end
initialize queue Q with all TSP sequences whose travelling time $< UB$, sorted in ascending order of travelling times;
initialize *visited archive* ;
repeat
    $current_seq \leftarrow Q.dequeue()$;
    if *travelling time of* $current_seq > LB$ **then**
        $LB \leftarrow$ travelling time of $current_seq$;
    end
    $current_sol \leftarrow$ solve subproblem 2 to obtain the corresponding ITSP solution of $current_seq$;
    if *number of processing-steps of* $current_sol >$ *number of processing-steps of* $bestsol$ **then**
        $newsol \leftarrow$ solve subproblem 1 with new number of processing-steps ;
        if $newsol$ *is better than* $bestsol$ **then**
            update $bestsol$ and UB ;
        end
    end
    if *stopping branching condition (according to lemma 1) is not satisfied for* $current_sol$ **then**
        $list_sequences \leftarrow$ create list of all children sequences of $current_seq$;
        foreach seq *in* $list_sequences$ **do**
            if seq *has not been visited* **then**
                $Q.enqueue(seq)$;
                update *visited archive* ;
            end
        end
    end
until Q *is emptied or gap closed*;

described in algorithm 3. A recent survey on the VNS can be found in (Hansen et al., 2010). The VNS has been successfully applied to many variants of the TSP (Meng et al., 2018, Sousa et al., 2015) and Vehicle Routing Problems (Pop et al., 2014, Wen et al., 2011).

Algorithm 3: The VNS framework

input : A set of neighborhood structures $N_k, k = 1, .., k_{max}$

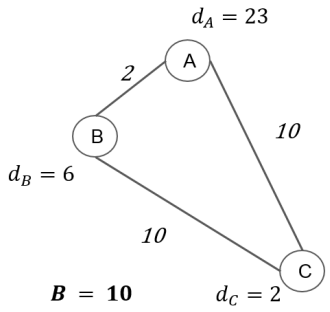
$x \leftarrow$ construct the initial solution;
repeat
  $k \leftarrow 1$;
  repeat
    Perturbation. Generate x' randomly from $N_k(x)$;
    Local search. $x'' \leftarrow$ apply some local search with x' as the initial solution;
    Neighborhood change. If x'' is better than x, set $x \leftarrow x''$, $(k \leftarrow 1)$ and continue the search with N_1; otherwise, set $k \leftarrow k + 1$;
  until $k = k_{max}$;
until *stopping condition is met*;

In this chapter, we use the general VNS (GVNS) (Hansen and Mladenović, 2002, Hansen et al., 2008), a variant of the VNS where the Variable Neighborhood Descent (VND) is used as the local search routine. The usual strategy for exploration of several neighborhoods within deterministic local search is sequential (Seq-VND) (Hansen et al., 2008), where a set of neighborhoods are used in the search. Local searches are performed in a pre-defined order of this set. Every time a better solution is found, the search returns to the first neighborhood in the list. The search stops when no improvement is found by any neighbourhoods.

In the following sections, we define the necessary components to apply the GVNS for the ITSP: initial solution constructor, local search procedure and perturbation procedure.

4.4.1 Solution encoding

Before describing the VNS components, we hereby present the solution representation. A solution is encoded as a list of processing-steps. Each processing-step j consists of information of the corresponding vertex's id (v), processing time (p_j) and waiting time (w_j) and the time that the processing actually starts (t_j), i.e. the arrival time plus waiting time. To speed up the calculation, for each vertex, a list of corresponding processing-steps is also recorded. The objective value of a solution is the total processing time, i.e., the sum of the starting time and the processing time of the last processing-step. An illustration of a solution encoding can be found in figure 4.3.



Processing -step j	0	1	2	3	4
Vertex v	A	B	A	A	C
p_j	10	6	10	3	2
w_j	0	0	0	3	0
q_j	0	0	0	10	0
t_j	0	12	20	33	46

$S = p_4 + t_4 = 48$

Figure 4.3: Solution encoding for an instance with 3 vertices. The solution includes 5 processing-steps where vertex A is visited three times.

4.4.2 Initial solution constructor

Two constructive heuristics are proposed, namely *NearestVertex* and *LongestVertex*.

NearestVertex This heuristic starts from a random vertex, progressively processes the current vertex maximally (e.g. until the maximum temperature is reached) and walks to a random vertex in a list of nearest vertices where processing can start before the cooling has stopped on the current one. If no such vertex exists, the processing is continued at the same vertex after a proper waiting time. The algorithm is described in algorithm 4.

LongestVertex The heuristic picks a random vertex from the list of α vertices with the longest remaining processing time. The processing of this vertex is combined with the complete processing of other vertices, in order to facilitate the cooldown of the current vertex, until it has been fully processed. The heuristic is described in algorithm 5.

4.4.3 Local search

The local search procedure we choose is the Sequential VND (Hansen et al., 2008), where a set of neighborhoods is used. The Sequential VND iteratively selects a move and calls the corresponding first improvement hill climbing local search to obtain a

Algorithm 4: NearestVertex

input : Instance ins

$current \leftarrow$ initialize solution with a random vertex ;
repeat
  maximally process $current$;
  $set_nearby \leftarrow$ list of vertices where processing can start before the cooling has stopped on $current$, ordered decreasingly by their processing time left;
  if set_nearby *is not empty* **then**
    $current \leftarrow$ choose a vertex randomly from the α first vertices of set_nearby;
  else
    $\rho \leftarrow$ processing time left on $current$;
    if $\rho \geq B$ **then**
      wait at $current$ for B time units;
    else
      wait at $current$ for ρ time units;
    end
  end
until *there is no unprocessed vertex left*;

Algorithm 5: LongestVertex

input : Instance ins

repeat
  $list_vertices \leftarrow$ list of vertices ordered decreasingly by their processing time left;
  $current \leftarrow$ choose a vertex randomly from α first vertices of $list_vertices$;
  repeat
    maximally process $current$;
    repeat
      $next \leftarrow$ find the nearest unprocessed vertex that can be finished processing before $current$ cools down completely ;
      maximally process $next$;
    until $current$ *is completely cooled down*;
  until $current$ *is completely processed*;
until *there is no unprocessed vertex left*;

local optimum which is then used as the starting point for the next iteration. The search terminates when no improvement is found by any neighborhood moves.

Thanks to the similarity in TSP and ITSP solution structures, neighborhood classes from the TSP are applicable for the ITSP. The following moves are used in our algorithm: 2-opt (Croes, 1958), 3-opt (Bock, 1958, Lin, 1965), or-opt and remove-and-reinsert (Nilsson, 2003).

Due to the temperature constraints, evaluating the benefit of a move is costly. Therefore, we propose to estimate an ITSP move before actually proceeding with the move. We illustrate this estimation using the 2-opt move as follows. Consider the TSP 2-opt move where edges $(j, j+1)$ and $(k-1, k)$ are replaced by $(j, k-1)$ and $(j+1, k)$ and the segment $(j+1, k-1)$ is reversed (see figure 4.4), the move yields a benefit if $c_{j,j+1} + c_{k-1,k} - (c_{j,k-1} + c_{j+1,k}) > 0$ (in symmetric instances). Therefore, checking if a move is beneficial in the TSP is $O(1)$.

In the ITSP, due to the temperature constraints, arrival temperatures, starting times and waiting times of all processing-steps starting from $k-1$ have to be recalculated. The new objective value is estimated by only considering processing-steps within an *estimated radius* r around j and k. The rationale for this estimation is that if there is a change in visiting times due to the temperature constraints, it is most likely caused by processing-steps in a small radius around the processing-steps involved in the move, since processing-steps of the same vertices tend to be grouped together in a small segment. The estimation and the reasoning for it is illustrated in figure 4.4 where $j = 3$, $k = 10$ and the move reverses the segment from 4 to 9. In the new solution, time information (starting time and waiting time) of processing-steps starting from 9 must be recalculated. To quickly estimate the move, only time information of the segment $(k-1, k-r)$ (processing-steps 9 and 8 in the example) are recalculated. The time difference between the old solution and the new solution of $k-r$ is forwarded to processing-step k (processing-step 10 in the example). Time information of processing-steps in segment $(k, k+r)$ (processing-steps $10, 11, 12$) is then calculated, taking into account the forwarded time difference. The difference between the starting time of processing-step $k+r$ in the old solution and $k+r$ in the new solution (processing-step 12 in the example) will give an estimated delta objective to evaluate the move.

To calculate the new time information of processing-step 9 in the new sequence, we need to look back to the processing-steps before it, i.e. processing-steps starting from 3 backwards to the beginning of the sequence, to check if temperature constraints are violated. In the estimation, we look back only $r = 2$ processing-steps before 3. As one can observe, the larger r is, the more precise the estimation becomes.

If the estimated objective value is smaller than the original one, i.e. the move might be beneficial, we proceed with the move and recalculate the whole solution to get the exact objective value. A similar scheme is applied for other moves.

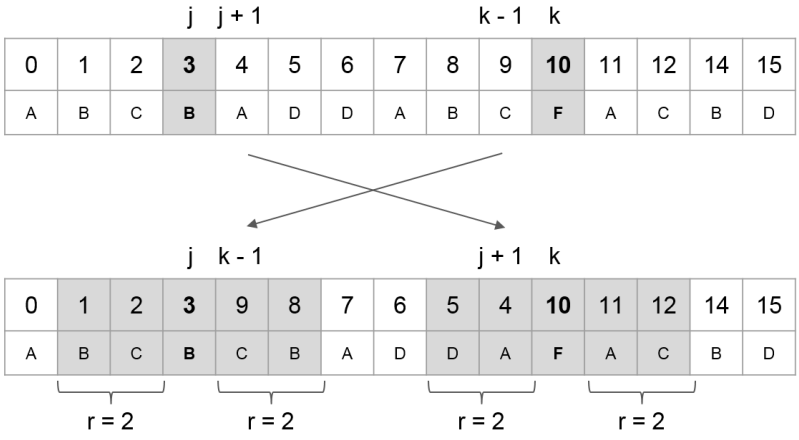


Figure 4.4: Move evaluation example.

For each neighborhood move, a first improvement hill climbing search is developed. When a local optimum is reached, the processing time of each processing-step is probably not optimal for the given sequence anymore. Therefore, a post processing is applied to adjust the processing time of each processing-step. This is done by solving the LP model in section 4.3.1 for the corresponding sequence.

4.4.4 Perturbation

Perturbation plays an important role in guiding the search to exploit new regions of the search space. Starting at the local optimum given by the local search procedure, perturbation makes a large move to jump to a distant neighbor in order to escape from the corresponding valley. We propose two perturbation moves for the framework: ruin-and-recreate and modified hyperedges exchange.

Ruin-and-Recreate (RR) A *RR* move (Schrimpf et al., 2000) includes two phases: the ruin phase removes some vertices from the tour while the recreating phase inserts those vertices into the tour. The *RR* for the ITSP is conducted as follows: In the ruin phase, a seed vertex chosen randomly is removed from the solution. Then a number of its closest vertices are also removed. Afterward, the recreate phase reinserts those vertices back into the tour using the best insertion strategy. The parameter of the RR move is the percentage of vertices being removed from the route.

Guided hyperedges exchange A hyperedge $H = (a_0, h)$ includes h consecutive edges and starts from vertex a_0. In hyperedges exchange, two hyperedges $H_1 = (a_0, h)$ and $H_2 = (b_0, h)$ are removed from the tour leaving their vertices isolated. Those vertices are then reconnected to the tour randomly. The hyperedges exchange move is illustrated in figure 4.5.

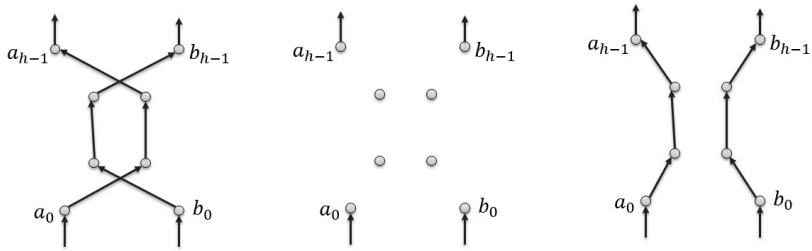


Figure 4.5: Hyperedges exchange.

In order to guide the algorithm to a promising region of the search space, we adopt the guided perturbation of (Burke et al., 2001) in the ITSP by utilizing information of the current tour. The quality of a hyperedge $H = (a_0, a_1, .., a_{h-1})$ is evaluated by:

$$\sum_{j \in H}(c_{j,j+1} - min_{v \in V | v \neq v_j} c_{v_j,v}) + \mu \sum_{j \in H} w_j$$

with w_j the waiting time at processing-step j and μ the normalizing factor. The first term of the formula measures the difference between the total distance from each visit to its successor in the hyperedge and to its closest neighbor. A large difference indicates that there might be a chance to obtain a shorter path by destroying and re-routing that path. The second aggregates the waiting times along the hyperedge. The reasoning behind this term is that if we have to wait for a long time at a vertex, then it might be better to visit other vertices instead of waiting there. The role of μ is to distribute the influence of the two terms equally. The value of μ is set as the ratio between the first term and the second term of the first hyperedge whose aggregate waiting time is positive (denoted by H^*).

$$\mu = \frac{\sum_{j \in H^*}(c_{j,j+1} - min_{v \in V | v \neq v_j} c_{v_j,v})}{\sum_{j \in H^*} w_j}$$

In total, a high value of the function implies a poor quality hyperedge.

In guided hyperedges exchange, hyperedges are sorted increasingly by their quality values. At each perturbation step, a hyperedge is chosen randomly from the first θ percent of hyperedges in the list while the second hyperedge is chosen completely randomly.

Algorithm 6: GVNS for the ITSP

$x \leftarrow$ construct the initial solution using *LongestVertex or NearestVertex*;
$k \leftarrow \Lambda_{min}$;
repeat
  Perturbation. $x' \leftarrow$ perturb x with strength k;
  Local search. $x'' \leftarrow$ apply VND with x' as the starting point;
  Neighborhood change.
  if *If* x'' *is better than* x **then**
    set $x \leftarrow x''$;
    continue the search with $k \leftarrow \Lambda_{min}$;
  else
    set $k \leftarrow k + \delta_\Lambda$;
  end
until $k \leq \Lambda_{max}$ *or a timeout is reached*;

4.4.5 The GVNS for the ITSP

Given the components described earlier, the GVNS for the ITSP is conducted as follows. The neighborhood change is given by varying the strength of perturbation, i.e. the size of hyper-edges in the perturbation step. Perturbation strength k is initialized with its minimum value Λ_{min}. At each iteration, a random solution is generated by perturbing the current solution with strength k. This solution serves as the starting point for the Seq-VND procedure. If a better solution is found, the incumbent solution is updated and k is reset to Λ_{min}. Otherwise, the solution is perturbed again with the same strength. After n perturbations without improvement, where n is the number of vertices of the instance, k is increased with a step of δ_Λ. The search terminates when k exceeds its maximum allowed value Λ_{max} or a timeout is reached. Our GVNS algorithm for the ITSP is described in algorithm 6.

Given two construction heuristics and two perturbation types, four different VNS metaheuristics are created. The two construction heuristics are named *Nearest* (NearestVertex) and *Longest* (LongestVertex) while the two perturbation types are named *RR* (Ruin-and-Recreate) and *Exchange* (guided hyperedges exchange). Their combinations result in four different VNS metaheuristics: *NearestRR*, *NearestExchange*, *LongestRR* and *LongestExchange* respectively.

4.5 Experiment

The experiment consists of two parts. In the first part, we evaluate the performance of the branch-and-bound (BNB) approach and analyse problem difficulty regarding their settings using small instances. In the second part, the VNS approach is evaluated on all instances. All algorithms are implemented in Java. The MIP solver used is Gurobi 6.0.5 (Gurobi Optimization, 2015). Tests for the BNB algorithm are carried out on a Core i7, 2.9 GHz Intel machine with 7.7 GB of RAM. Tests for the VNS are carried out on a Linux-based PC cluster, with Intel Xeon E5-2680v3 2.5 GHz CPUs and a budget of 9GB of RAM for each core.

4.5.1 Test instances

Since the problem is new in literature, we introduce a problem library consisting of two sets: set R includes instances generated randomly and set T includes instances generated from instances of the TSPLIB (Reinelt, 1991). All instances and best results are made publicly available at https://set.kuleuven.be/codes/itsp_page.

There are two important features influencing the difficulty of an instance: (1) *bound ratio* which is the ratio between the maximum temperature (B) and the average travelling time between vertices ($\bar{C}$). (2) instance *density* which determines the number of vertices of each type. In our test instances, there are 4 types of vertices: 1, 2, 3 and 4-steps. A γ-$step$ vertex needs at least γ processing-steps to finish, i.e. its processing time p lies in the range $(\gamma - 1)B < p \leq \gamma B$. The number of vertices of each type in an instance is determined by its density setting. There are in total 11 density settings from *D0* to *D10*. Lower density setting results in sparser instances. The detailed percentages of each type of vertex in each density setting are given in table 4.1.

In set *R*, vertex coordinates are generated randomly in a square box. In set *T*, vertex coordinates are taken from the corresponding TSPLIB instance. Then, given a bound ratio and a density setting, processing times of vertices are generated accordingly. Instances are named after their size and settings. For example, an instance with size of *20* vertices, density setting *D0* and bound ratio 10 is named 20_D0_10. In case several instances have the same setting, an index is appended at the end of their names, e.g. 20_D0_10_1 and 20_D0_10_2. The names of each instance in set *T* includes the name of the corresponding instance in the TSPLIB. E.g. instance 77_D9_10_pr76 is generated from instance *pr76*, consists of 77 vertices (including the depot), has density setting $D9$ and bound ratio 10. Similarly to set R, an index is used to distinguish instances generated from the same TSPLIB instance and sharing the same setting. Both set R and T include many subsets for different experiments which will be presented in more detail in the following sessions.

Table 4.1: Density setting configuration.

	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
single-step (%)	100	90	80	70	60	50	40	30	20	10	10
2-steps (%)	0	10	20	30	40	40	50	60	70	70	70
3-steps (%)	0	0	0	0	0	10	10	10	10	20	10
4-steps (%)	0	0	0	0	0	0	0	0	0	0	10

Table 4.2: Instance configuration of set R.

Subset	No. instances	Instance size	Bound ratio	Density
R1	154	6,7,8,9,10,11,12	1,100	D0 to D10
R2	200	10	0.1, 0.2, 0.4, ..., 51.2 (10 settings)	D2
R3	220	10	10	D0 to D10

4.5.2 Evaluation of the branch-and-bound approach

Three different experiments are carried out for different purposes. Three subsets of instances (R1, R2 and R3) with different sizes and settings are generated consequently.

In the first experiment on subset R1, the BNB performance with different instance sizes is evaluated. Instance size ranges from 6 to 12 vertices. For each size, 22 instances are generated corresponding to the combination of 11 density settings and 2 bound ratio settings (1 and 100).

In the second experiment on subset R2, we aim to analyse problem difficulty regarding *bound ratio* setting. Test instances have *bound ratio* ranging from 0.1 to 51.2, the next ratio is double the current ratio $[0.1, 0.2, 0.4, ...25.6, 51.2]$. Instance size and density setting are fixed as 10 vertices and *D2* respectively for all instances. For each bound ratio, 20 instances are generated.

In the third experiment on subset R3, we analyse problem difficulty regarding density setting. There are 11 density settings as listed in table 4.1. Instance size is fixed as 10 vertices and *bound ratio* is 10 for all instances. For each density setting, 20 instances are generated. The summary of three subsets and their configurations can be found in table 4.2.

A time limit of 1200 seconds is imposed on the MIP solver to solve subproblems and 3600 seconds on the BNB algorithm.

The results of this experiment are analysed in figure 4.6 and table 4.3. Figure 4.6 shows the box plots of the running times of the algorithm on the three subsets while table 4.3 reports the average values of the running times, initializing times and the size of the queue for each subset, categorized by the corresponding targeted feature (instance

size in R1, bound ratio in R2 and density setting in R3). In most of the test instances, optimal solutions are found, except some large instances in set *R1*.

Figure 4.6a shows the results for experiment 1 on subset *R1* of instances with different sizes. We can see easily from the figure that problem difficulty depends highly on instance size (which is predictable). The corresponding initializing times and queue sizes given in table 4.3 show that the larger an instance is, the longer it takes to initialize the queue. This is caused by the exponential increase in the number of TSP sequences when the instance size increases. This suggests that the BNB approach is not scalable for big instances.

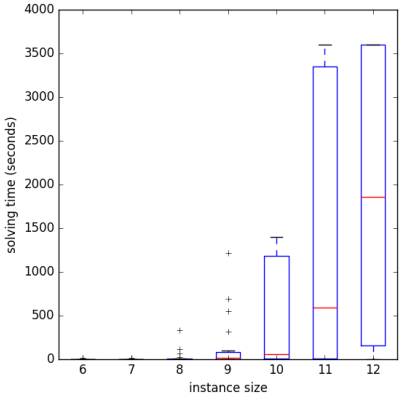
Figure 4.6b shows the results for experiment 2 on subset *R2* including instances with the same size and density setting but different bound ratio. As shown in the figure, problem difficulty increases when bound ratio increases. This can be explained as follows. A large bound ratio means a large difference between the maximum temperature B and the average travelling time $\bar{C}$, which leads to more waiting times in the solution. Since LB is the total travelling time of the solution and UB is the sum of the total travelling time and the aggregate waiting time, a large aggregate waiting time results in a large gap between LB and UB. A large gap makes an instance difficult for the BNB algorithm to solve. This also results in a bad cut-off value for the initializing process, which is reflected in the large initializing times and queue sizes in instances with large bound ratio in table 4.3. This explains why instances with high bound ratio settings tend to be more difficult. This also explains the small differences when the bound ratio exceeds 2. The larger the bound ratio is, the larger the aggregate waiting time in the solution is. However, when the bound ratio exceeds 2, the waiting times are mostly 0 and cannot go any lower. Therefore, given the same instance size, instances with bound ratio exceeds 2 have similar level of difficulty.

Figure 4.6c shows the results for experiment 3 on subset *R3* which aims to compare the BNB performance on different density settings. Predictably, the denser an instance is, the harder it is to solve. This can also be explained by the large aggregate waiting time in dense instances.

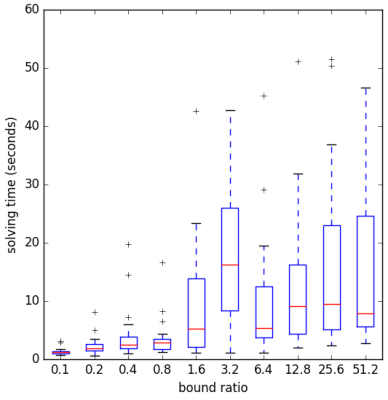
To give some more insight into how the BNB works, we analyse the detailed results of some selected instances from all 3 subsets in table 4.4. Information reported includes: instance name, itsp size (e.g. minimum number of processing-steps), search status ($FINISHED$ means optimum found, the queue is emptied within the time limit or the gap is closed), running time, queue initializing time, size of the search queue, lower bound, upper bound and finally solution gap. What stands out in the table is that in *D0* instances, the number of nodes in the search tree is 0. This can be explained as follows. For *D0* instances where all vertices are *1-step* vertices, a TSP solution makes a valid ITSP solution. Therefore, in these instances, the gaps are closed right at the initial phase when the lower bound and upper bound are obtained. That is why those instances are easy to solve, resulting in small solving times of less than 2 seconds. By contrast, as

Table 4.3: Branch-and-bound results on the three subsets *R1*, *R2* and *R3* respectively.

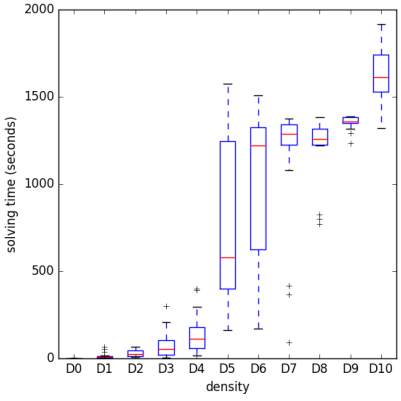
	Running time		Initializing time		Queue size	
	Avg (seconds)	Std (%)	Avg (seconds)	Std (%)	Avg (#sequences)	Std (%)
Instance size (R1)						
6	1.56	247.41	0.05	55.16	47.36	92.58
7	2.11	180.45	0.17	76.86	136.91	115.02
8	27.59	262.59	0.92	96.76	989.55	148.07
9	146.54	201.74	5.07	114.18	4763.86	170.44
10	453.66	127.03	47.01	142.99	58412	176.01
11	1366.57	111.49	1096.72	142.4	196706.68	162.28
12	1982.27	79.28	1845.38	89.4	44230.86	194.8
Bound ratio (R2)						
0.1	1.39	43.54	0.5	86.92	9.3	83.92
0.2	2.38	68.44	0.99	106	41.95	94.21
0.4	4.22	110.14	1.54	95.92	159.8	159.98
0.8	3.66	93.09	1.11	86.14	125.5	181.24
1.6	9.14	108.87	3.65	131.55	849.6	141.38
3.2	17.97	64.99	7.33	102.9	1345.78	205.2
6.4	10.3	105.59	4.89	127.75	830.26	185.93
12.8	12.54	97.3	5.93	137.03	1901.35	172.96
25.6	16.34	92.38	6.02	112.16	1958.75	163.95
51.2	16.02	91.12	7.82	129.12	1929.05	171.5
Density setting (R3)						
D0	0.85	57.41	0	0	0	0
D1	14.82	116.68	7.6	130.8	620.9	208.15
D2	29.2	64.76	11.44	80.16	1002.95	134.64
D3	81.44	98.12	42.63	146.54	10902.8	172.91
D4	135.93	81.11	45.14	107.24	12652.8	167.87
D5	778.6	58.78	133.93	70.58	57030.1	103.95
D6	1003.41	43.59	120.4	79.78	89603.8	95.05
D7	1142.34	32.23	96.66	62.74	128895.2	86.57
D8	1212.93	14.93	78.84	67.38	94191.5	100.81
D9	1350.65	3	152.95	22.23	232340.47	38.31
D10	1615.17	9.93	414.89	38.63	275203.7	37.05



(a) Comparison on instance size, subset R1.



(b) Comparison on bound ratio, subset R2.



(c) Comparison on density setting, subset R3.

Figure 4.6: Branch-and-bound performance on different subsets.

Table 4.4: Detailed results on some selected instances in set *R*.

Instance	Itspsize	Status	Running time	Queue initializing time	Queue size	LB	UB	Gap
8_D9_100	16	FINISHED	466.67	2.56	4974	476	476	0.00%
8_D9_1	16	FINISHED	156.72	2.36	3084	406	406	0.00%
9_D0_100	10	FINISHED	0.15	0	0	334	334	0.00%
9_D0_1	10	FINISHED	0.14	0	0	285	285	0.00%
10_D1_1	12	FINISHED	2.95	1.36	128	412	412	0.00%
10_D2_100	13	FINISHED	10.23	4.44	564	387	387	0.00%
10_D2_51.2_10	13	FINISHED	39.2	18.7	5328	488	488	0.00%
10_D2_51.2_11	13	FINISHED	5.72	1.94	318	361	361	0.00%
10_D6_10_5	10	FINISHED	172.67	71.22	57194	502	502	0.00%
10_D6_10_6	10	FINISHED	663.03	119.96	121862	498	498	0.00%
11_D3_100	15	FINISHED	31.5	5.05	452	356	356	0.00%
11_D3_1	15	FINISHED	4.17	0.99	191	315	315	0.00%
12_D0_1	13	FINISHED	1.47	0	0	422	422	0.00%
12_D4_100	17	FINISHED	2605.94	711.32	50962	486	486	0.00%
12_D4_1	17	FINISHED	443.75	129.63	2128	384	384	0.00%
12_D5_100	19	TIMEOUT	3600	3600	15611	348	598	41.81%
12_D5_1	19	TIMEOUT	3600	1255.68	220195	298	416	28.37%
12_D6_100	21	TIMEOUT	3600	3600	2734	395	633	37.60%
12_D10_100	26	TIMEOUT	3600	3600	2475	376	845	55.50%

the queue size is exponential of the instance size, in larger instances the time taken to initialize the queue takes up all the time budget. This is the main reason why the BNB is not scalable to big instances.

Table 4.5: Parameters tuning.

Parameter	Description	Test values	Best setting
α	Construction heuristics' list length	10, 50 (%)	10
r	Estimated radius for local search	3, 10 (processing-steps)	3
θ	Hyperedges list size proportion	25, 75 (%)	75
$(\Lambda_{min}, \Lambda_{max})$	Perturbation min strength and max strength	(10, 60), (30, 90) (%)	(30, 90)
δ_Λ	Perturbation strength step	5, 10 (%)	10

4.5.3 Calibration of the metaheuristics

In order to tune the metaheuristic parameters, a tuning set is created by selecting instances from all subsets of set R and T in such a way that instances of all sizes and configurations are used. The size of the tuning set is 50 instances. For each parameter, 2 values are selected in a way that they cover most of its value ranges. The combination of 5 sets of parameters, with 2 values for each, results in 32 experiments. Note that θ, the hyperedge list size proportion, is used in *NearestExchange* and *LongestExchange* only. For each instance, each experiment, the four metaheuristics (*NearestRR*, *NearestExchange*, *LongestRR* and *LongestExchange*) are applied for 10 runs with a time limit of 100 seconds for each. The best setting is selected from the combination which gives the smallest average deviation from best solutions. However, the one-way ANOVA test performed on the results shows that there is no significant difference between different parameter settings ($F = 0.08, p = 0.99$). The parameters, their tested values and the best settings are listed in table 4.5.

4.5.4 Results and metaheuristics comparison

To evaluate and compare the four metaheuristics proposed in this chapter, three experiments are conducted. The results of all experiments in this section are the average of 5 runs.

In the first experiment, the four metaheuristics are tested on all three subsets of set *R* which consists of small randomly generated instances. A time limit of 5 seconds is imposed on all of them. Table 4.6 shows the average deviation from the optimal solution for each metaheuristic and instance set. It can be seen from the table that all metaheuristics perform very well on those instances, especially *NearestRR* and *NearestExchange* with the average deviation of 0% in all instance sets. *LongestRR* and *LongestExchange* show small deviations especially in set *R2* but is still less than 0.2%.

For the second experiment, a set of special instances $T1$ are generated from the TSPLIB in a way that the optimal solutions are known. From a TSPLIB instance, a special instance is generated by setting the maximum temperature equal to 100, and the

Table 4.6: The VNS performance on set *R*.

	NearestRR	NearestExchange	LongestRR	LongestExchange
R1	0.00%	0.00%	0.08%	0.08%
R2	0.00%	0.00%	0.16%	0.17%
R3	0.00%	0.00%	0.01%	0.01%

processing time of each vertex equal to 1. In such instances, each vertex is visited exactly once. Therefore its optimal solution is the optimal TSP solution (which is known) plus the number of vertices. $T1$ includes 80 instances corresponding with 80 TSPLIB instances with the size ranging from 14 to 3028 vertices. The time limit is set to 1800 seconds. The results for the second experiment are shown in figure 4.7. The first plot shows the deviation of the metaheuristics' results from the optimal solutions while the running time is showed in the second plot. The results are good on *NearestRR* and *NearestExchange* with average optimality gaps of 4.09 and 4.21 respectively while *LongestRR* and *LongestExchange*, with average gaps 13.39 and 14.14 respectively, reveal inefficiency in large instances.

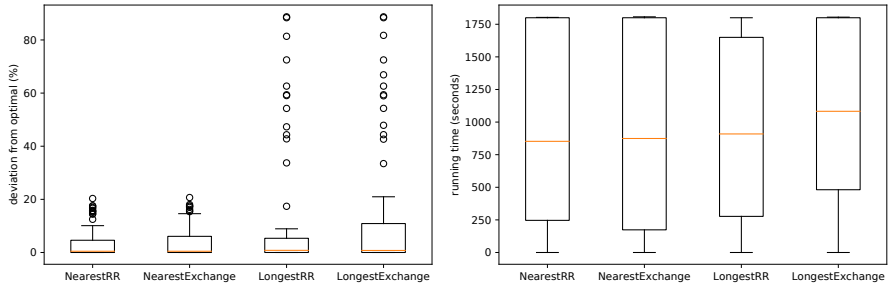


Figure 4.7: Results on special instances of set *T1*.

In the third experiment, the four metaheuristics are tested on big instances of set *T2*. 30 instances from the TSPLIB are randomly selected to generate instances in this set. For each instance, we generate 10 instances corresponding to the combination of density settings $D0, D2, D4, D7, D9$ and bound ratio settings $0.1, 10$. Therefore, set *T2* consists of 300 instances. The sizes of instances in set *T2* vary from 49 to 5916 vertices and the number of minimum processing-steps (which is called the ITSP-size) ranges from 49 to 12422. Similarly to the second experiment, a time limit of 1800 seconds is imposed.

Since optimal solutions are unknown for those instances, we compare the results with the best known solutions obtained so far for each instance. In table 4.7, the results are reported by groups of instances. Reported information includes the average deviation

Table 4.7: Metaheuristics performance on set *T2*.

	NearestRR		NearestExchange		LongestRR		LongestExchange	
	Avg. dev (%)	Avg. Time (seconds)	Avg. dev (%)	Avg. Time (seconds)	Avg. dev (%)	Avg. Time (seconds)	Avg. dev (%)	Avg. Time (seconds)
Overall	**0.35**	1187.52	0.76	1270.66	7.31	1266.16	7.95	1229.58
Density								
D0	**0.48**	949.49	1.24	1003.1	6.04	1040.01	7.01	985.58
D2	**0.44**	1197.03	0.91	1284.02	7.83	1299.87	8.47	1187.41
D4	**0.28**	1214.66	0.63	1288.11	6.96	1284.55	7.66	1269.08
D7	**0.28**	1271.36	0.51	1381.5	7.37	1384.1	7.87	1358.81
D9	**0.24**	1334.46	0.48	1429.08	8.55	1348.56	8.9	1373.51
Bound ratio								
0.1	**0.68**	1224.63	1.49	1249.34	14.51	1272.89	15.76	1245.03
10	**0.03**	1151.44	0.06	1291.39	0.31	1259.62	0.35	1214.55

from the best know solutions and average running time. The first row of the table shows the results of each metaheuristic on the whole set. The next four rows show the detailed performances of each metaheuristic on each subgroup of set *T* corresponding to 5 different density settings ($D0$ to $D9$). The last two lines show results on 2 subsets corresponding to 2 bound ratio settings (0.1 and 10). The results diverge more on instances with a small bound ratio.

An one-way ANOVA test on the results shows a statistically significant difference between the four algorithms ($F = 43.52, p < 0.001$). It can be seen that *NearestRR* performs the best among the four metaheuristics. Post-hoc paired-samples $t - tests$ between *NearestRR* and the other three metaheuristics all give p-value less than 0.001. Applying Bonferroni correction with the number of tests $n = 3$, we have $\alpha_{altered} = 0.05/3 = 0.016$. Since the p-values of all three post-hoc tests are less than $\alpha_{altered}$, we conclude that *NearestRR* outperforms the other three metaheuristics.

The metaheuristics using *NearestVertex* outperform the ones using *LongestVertex* in general, especially in a small bound ratio setting (0.1). This can be explained by the inclusion of the *NearestVertex* starting solution. This leads to a better performance, because it is more worthwhile to move to vertices that are close by, rather than to move to vertices which have the longest remaining processing time. This holds especially for higher density settings, since in this case more visits to each vertex are required, resulting in higher travelling times for *LongestVertex*. Furthermore, Ruin-and-Recreate outperforms Hyperedges Exchange, which we suspect is resulted from the fact that the former works with a closely connected subset of vertices, whereas the latter considers longer edges. Hence, it is shown that also for perturbation it is better to change the solution by considering vertices closest to those removed (ruin phase) rather than employing hyperedges. This results in a comparatively better performance of *NearestRR* and *LongestRR* versus *NearestExchange* and *LongestExchange* respectively.

4.6 Conclusion and future work

In this chapter we introduced a new variant of the TSP, namely the Intermittent Travelling Salesman Problem, where a vertex may have to be visited more than once and there is a time lapse enforced between two consecutive visits due to the temperature constraint. An exact branch-and-bound approach is proposed to solve small instances. Instance difficulty regarding its properties is analysed based on the results from the exact approach. To tackle large instances, four VNS metaheuristics are built and compared with each other. An instance library consisting of two sets is built and made publicly available. The first set R contains small, randomly generated instances while the second set T contains large instances generated from instances of the TSPLIB. The results on small instances illustrate that all four metaheuristics result in very small deviations from optimal solutions, especially *NearestRR* and *NearestExchange*. On big instances, *NearestRR* performs the best on all instance sets while the performances of the other metaheuristics depend on instance properties.

The problem described in this chapter is a simplification with respect to the real world setting in that we approximated the cooling schemes by linear functions and supposed them to be uniform over the work-piece. In future work, quadratic or exponential cooling and heating functions will be studied, as well as the influence of position dependent cooling and the interaction between nearby vertices. The simplified problem in its own right, even though it is shown to be accessible through usual combinatorial optimisation approaches, turns out to be particularly challenging. Interesting subjects to investigate in the future are stronger exact approaches, e.g. based on decomposition, for - approximately - solving larger instances. Heuristic approaches relying on alternative problem representations or approximate delta evaluation of the objective function also deserve further attention.

Chapter 5

The Radiotherapy Scheduling Problem

Goal of the chapter Radiotherapy treatment is a popular form of cancer treatment where a patient receives a large amount of radiation over a course of several consecutive days. In the most common form of radiotherapy, the radiation is delivered by a linear accelerator (linac). Planning the usage of linacs is essential in reducing the waiting time of patients, which directly affects the treatment outcomes. We study the Radiotherapy Scheduling Problem, which consists of finding the treatment schedule for a set of patients given a set of linacs respecting many side constraints and preferences. We propose a two-phase approach for the problem. In the first phase, patient treatments are assigned to specific linacs and days. The second phase then decides the sequence of patients on each day/linac and the specific appointment times. For the first phase, a Mixed Integer Linear Programming (MIP) model is proposed and solved using CPLEX. For the second phase, a MIP and a Constraint Programming (CP) model are proposed. The test data is generated based on real data from the cancer center of CHUM, a large cancer center in Montréal, Canada, with about 3500 new patients and 40.000 treatments a year. The results show that the CP approach outperforms MIP for this problem and is able to solve real-sized instances.

This work was conducted partly during my internship in Montréal, Canada from April to August, 2019. The content of this chapter has been accepted as a full paper at MISTA - the Multidisciplinary International Scheduling Conference in Ningbo, China, December 2019.

- Tu-San Pham, Louis-Martin Rousseau, and Patrick De Causmaecker. The Radiotherapy Scheduling Problem. In *Proceedings of MISTA'2019*, 2019b.

5.1 Introduction

The number of cancer cases has been increasing sharply in the last few decades. According to the World Health Organization (WHO) [1], cancer is the second leading cause of death globally in 2018. It accounts for about 1 in 6 deaths. In particular, in Canada nearly 1 in 2 Canadians is expected to be diagnosed with cancer in their lifetime and about 1 in 4 Canadians to die from cancer (Can, 2018). The treatment facilities and personnel, however, do not grow proportionally, which not only causes lots of distress to the treatment facilities and staff but also results in long waiting for patients. The long waiting time (also referred as "killing time") of treatments having negative impact on the clinical outcomes has been proven through many articles (Chen et al., 2008, Coles et al., 2003, Mackillop, 2007). Therefore, better planning to reduce waiting times is crucial in improving cancer treatment results.

Before deciding a treatment plan, a set of tests and imaging need to be done on the patient, which might include x-rays, CT scans, MRI scans, PET scans, etc. The treatment team then carefully makes a treatment plan for each patient based on his/her diagnosis. There are many options for treatments, e.g. surgery, chemotherapy, radiotherapy. Sometimes different kinds of treatment are combined together on one patient and can be repeated as needed. This work focuses on the radiotherapy treatment (RT) where a patient receives a high dose of radiation to kill cancer cells. The radiation is usually divided into a number of smaller doses called **fractions** to allow healthy cells to recover between treatments. A series of fractions is delivered during a course of several consecutive days with breaks in the weekends. In the most common form of radiotherapy, the radiation is delivered by a *linear accelerator* (linac).

The waiting time of RT treatment depends highly on the planning of linac usage. According to Canadian Institute for Healthy Information, in 2018 50% of Canadian patients started treatments within 8 days, 90% started treatments within 20 days and 97% started treatments within the recommended maximum 28 days (see figure 5.1). Those numbers are getting worse in the recent years (see figure 5.2) due to the increasing number of patients. Better planning in treatments is hence vital to reduce the "killing time"

[1] https://www.who.int/news-room/fact-sheets/detail/cancer
[2] https://yourhealthsystem.cihi.ca
[3] Canadian Institute for Healthy Information

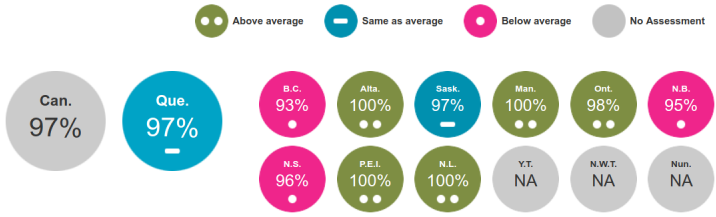


Figure 5.1: Percentages of patients receiving treatments within the recommended deadline by states in Canada (2018).[2]

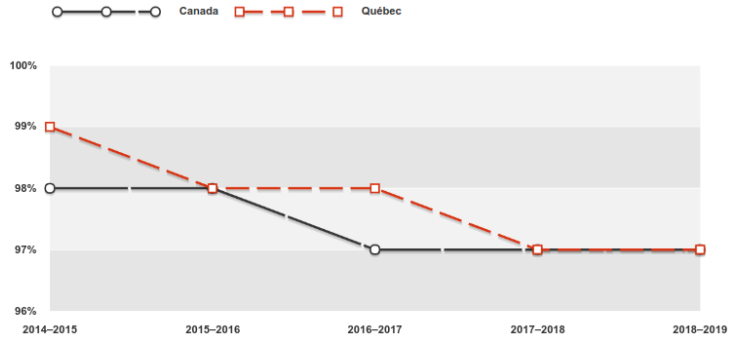


Figure 5.2: Percentages of patients receiving treatments before the recommended deadline in Quebec. [3]

In this chapter, we investigate the Radiotherapy Scheduling Problem (RTSP), which consists of deciding the radiotherapy treatment schedule for a set of patients in a horizon given a set of linacs. Many constraints have to be considered, e.g. patients' due days, linacs' capacity, patients' preferences on appointment times, consistency of treatment in terms of patient-linac assignments (i.e. a patient should not switch between linacs during his/her treatment period) and appointment times (i.e. a patient's appointment times should be consistent during his/her treatment period). Currently, in practice, most centers still create their appointment schedules by hand, which leaves lots of room for improvement. The problem, however, is complicated due to several strategies and constraints varying from center to center, which results in a arge number of variants of the RTSP. In this chapter, we consider a real-world variant of the RTSP present in the cancer center of CHUM (Centre hospitalier de l'Université de Montréal), a large cancer center in Montréal, Canada. Nevertheless, we aim to make the model as general and extensible as possible by taking into account many constraints and considerations.

The RTSP can be classified as a flexible job-shop problem with duplicate machines (Maccarthy and Liu, 1993). Each patient can be seen as a job consisting of several operations corresponding to his/her fractions, which must be scheduled in a specific order. However, some most important constraints and objectives of the RTSP such as consistency in patient-linac assignments, consistency in appointment times, or changes of fix patients are distinctive and not considered in the job-shop literature. Consequentially, it is difficult to apply traditional algorithms for job-shop problems to the RTSP. A wide variety of constraints and objectives differing from hospital to hospital makes the problem even more difficult. For this reason, most studies in literature apply heuristic methods. Exact algorithms either neglect or relax some objectives and constraints or only solve problems with a limited number of linacs and patients within a short time horizon. The first two mathematical models of the RTSP were proposed by Conforti et al. (2008). The objective of the models is to maximize the number of treated patients in a given horizon with the assumption that all treatments have a fixed duration of 15 minutes. The model is discrete with a fixed treatment duration of 15' assumed for all patients. The two models were tested on simple instances with 1 linac and 1 week horizon. The models were then extended in (Conforti et al., 2010) to tackle more realistic instances where the treatment times of patients vary. The test set consists of 2 linacs, a horizon of 1 weeks and 40 patients in the waiting list. In (Saure et al., 2012), the radiotherapy scheduling problem is modelled as a discounted Markov decision process to provide a dynamic policy to take into account future events. Legrain et al. (2015) developed a hybrid method combining stochastic optimization and online optimization to solve the online version of the problem with stochastic demands. All of the above studies, however, only assign patients to days, linacs and probably specific shifts while neglecting the order of patients on a specific linac or an exact appointment time of each patient. Vieira et al. (2019) proposed a mathematical programming model that can handle the time preferences of patients and sequences of patients on a specific linac. In (Sauré et al., 2020), the authors extended their previous work in 2012 to incorporate stochastic service times and perform a numerical analysis on a number of scenarios. To tackle large size instances, heuristics and metaheuristics are widely applied. Petrovic and Leite-Rocha (2008), Petrovic et al. (2006) proposed some constructive approaches and a GRASP-based algorithm. Kapamara and Petrovic (2009) proposed four heuristic methods to solve a problem which consists of both pre-treatment and treatment stages. Similarly, heuristic approaches are also used in (Maschler and Raidl, 2018, Maschler et al., 2016, Vogl et al., 2019). Each of the above-mentioned work, however, works with a different set of constraints and assumptions.

Aiming to solve larger instances and to provide a flexible model which can cover most popular constraints and objectives, we propose a two-phase approach for the problem. In the first phase, patient treatments are assigned to specific days and linacs. The second phase then decides the sequence of patients on each day/linac and the specific appointment times. For the first phase, a Mixed Integer Linear Programming model is

proposed and solved using CPLEX. For the second phase, a Constraint Programming model is proposed and solved using CPOptimizer. We generate realistic instances based on real data from CHUM.

The contribution of this chapter are three-fold. First, we introduce a general model, which is based on the model in (Vieira et al., 2019) but is extended with a wide range of constraints and objectives. Our model tackles many constraints that are not considered together in many previous works, for example, the consistency in patient-linac assignments, or the changes in appointment times of fixed patients from the previous planning periods. Second, to the best of our knowledge, this is the first time constraint programming is successfully applied to the RTSP. Third, we describe in this chapter how realistic instances are generated based on real data. In real life, the planning is usually made on a partly-full schedule which is filled with appointments from the previous planning periods. The schedule's layout and characteristics affect deeply the algorithm's performance. Hence it is important to test the algorithms on close-to realistic instances. Most works in literature either test their algorithms on empty instances with no fixed appointments or do not mention how their test instances are generated.

The structure of the chapter is as follows. The problem description along with the mathematical model are given in section 5.2. Section 5.3 presents a two-phase model for the problem while data descriptions and numerical results are presented in section 5.4. Section 5.5 closes the chapter with conclusions and future work.

5.2 Problem description

The radiotherapy scheduling problem (RTSP) consists of finding the schedule of treatments for a set of patients $\mathcal{P}$ over a planning horizon $\mathcal{T}$ given a set of linacs $\mathcal{K}$. Each patient $i \in \mathcal{P}$ has a release day r_i, which is the day that the patient is ready for treatment; a due day d_i, before which the treatment must start. A patient's due day is set based on his/her priority. Patients are categorized into palliative and curative. Palliative patients usually need urgent treatments within 1 to 3 days to relieve intense pain while curative patients can wait for longer to receive their treatments. Each patient i is associated with a treatment plan, which specifies the number of fractions I_i that the patient needs to receive, and the duration of each fraction (p_i) and the eligible linacs. All fractions of a patient have identical length, which is a multiple of 5, 10 or 15 minutes. In this chapter, we assume the faction duration is a multiple of 5 minutes and hence is measured in blocks of 5 minutes. A patient receives one fraction per day and the treatment must be carried out daily. Each treatment plan is also associated with one or several eligible groups of linac. Patients can switch between linacs of the same subgroup during the course of treatment. The total number of linacs assigned to a patient should

be minimized, due to several reasons: (1) each linac is usually associated with a set of technicians which will get familiar with patients' condition before their treatment; (2) some patient-specified set up might have to be performed before each treatment, which leads to more time wasted if the treatment is carried out in many different rooms. Each linac has a daily capacity of C time slots. Some patients have preferences for their appointment times, usually in the early morning or late in the afternoon. The preference of patient i is indicated by a time window $[t_i^{min}, t_i^{max}]$, within which the appointment should take place. For a patient's convenience, his/her appointment times should be consistent during the course of treatment. The time preference and appointment time consistency, however, only apply for curative patients. Finally, in practice, the fixed appointments of patients from the previous periods sometimes can be moved to make space for new patients, in case there are several small intermittent available slots on the linac. However, the changes in appointment should be minimized.

The problem has many objectives, among which the most important one is reducing the waiting time. As analyzed before, the earlier the treatment starts, the better it is for their condition. Therefore, the treatment should be scheduled as soon as possible once the patient is ready. Other objectives includes: patient-linac consistency; appointment time consistency; preference time window for curative patients; and finally changes in appointment times of fixed patients.

We hereby present a Mixed Integer Linear Programming (MIP) model for the RTSP.

Parameters:

- $\mathcal{P}$: the set of patients, which consists of two subsets of palliative ($\mathcal{P}^P$) and non-palliative patients ($\mathcal{P}^N$). $\mathcal{P}$ is also divided into two subsets of new patients ($\hat{\mathcal{P}}$) and fixed patients ($\bar{\mathcal{P}}$) from the previous planning period.
- $\mathcal{T}$: the set of days in the horizon.
- $\mathcal{S}$: the set of time slots of each day.
- $\mathcal{K}$: the set of linacs.
- $\mathcal{K}^i$: the set of eligible linacs of patient i.
- C_k^t: the capacity of linac k in day t.
- r_i: the release day of patient i.
- d_i: the due day of patient i.
- p_i: the fraction duration for patient i.
- I_i: the number of fractions of patient i.
- $[t_i^{min}, t_i^{max}]$: the preference time window of patient i.
- $\bar{s}_i^t$: the appointment time slot of the fixed patient i on day t from the previous period.

Variables:

- x_{is}^{tk}: binary variables, is equal to 1 if patient i is assigned to linac k at time slot s of day t.
- z_i^t: binary variables, is equal to 1 if patient i receives his/her first treatment on day t.
- w_{ik}: binary variables, is equal to 1 if patient i is assigned to linac k during his/her treatment.
- $\bar{\Delta}_i^t$ and $\underline{\Delta}_i^t$: continuous variables, represent deviation from the preference time window of the new, non-palliative patient $i \in \mathcal{P}^N$ on day t (sooner or later).
- $\bar{\theta}_i$ and $\underline{\theta}_i$: continuous variables, represent the earliest and latest appointment time slots of the new, non-palliative patients $i \in \mathcal{P}^N$.

- $\bar{\phi}_i^t$ and $\underline{\phi}_i^t$: continuous variables, represent changes in appointment times (to sooner and later) of the fixed patient $i \in \bar{\mathcal{P}}$ on day t.

Model:

$$\text{minimize } \omega_1 \sum_{i\in P}\sum_{t\in T} t z_i^t + \omega_2 \sum_{i\in P}\sum_{k\in K} w_{ik}$$

$$+ \omega_3 \sum_{i\in\mathcal{P}^N, t\in T} (\bar{\Delta}_{it} + \underline{\Delta}_{it}) + \omega_4 \sum_{i\in\mathcal{P}^N} (\bar{\theta}_i - \underline{\theta}_i) + \omega_5 \sum_{i\in\mathcal{P}, t\in T} (\bar{\phi}_i^t + \underline{\phi}_i^t) \quad (5.1)$$

subject to

$$\sum_{k\in\mathcal{K}}\sum_{s\in\mathcal{S}} x_{is}^{tk} \le 1, \qquad \forall i \in \mathcal{P}, t \in \mathcal{T} \quad (5.2)$$

$$\sum_{i\in\mathcal{P}} x_{is}^{tk} \le 1, \qquad \forall k \in \mathcal{K}, s \in \mathcal{S}, t \in \mathcal{T} \quad (5.3)$$

$$x_{is}^{tk} = 0, \qquad \forall i \in \mathcal{P}, t \in \mathcal{T}, k \notin \mathcal{K}^i, s \in \mathcal{S} \quad (5.4)$$

$$\sum_{k\in\mathcal{K}}\sum_{s\in\mathcal{S}}\sum_{t\in\mathcal{T}} x_{is}^{tk} \le I_i, \qquad \forall i \in \mathcal{P} \quad (5.5)$$

$$\sum_{k\in\mathcal{K}}\sum_{s\in\mathcal{S}} (x_{is}^{tk} - x_{is}^{t-1,k}) \le \sum_{k\in\mathcal{K}, s\in\mathcal{S}} x_{is}^{nk}, \qquad \forall i \in \mathcal{P}, t \in \mathcal{T},$$

$$n = t+1, \ldots, \min\{|\mathcal{T}| - 1, t + \mathcal{I}_i - 1\},$$

$$\{x_{is}^{-1,k} = 0, \forall i, k, s\} \quad (5.6)$$

$$\sum_{k\in\mathcal{K}}\sum_{s\in\mathcal{S}}\sum_{t=0}^{d_i} x_{is}^{tk} \ge 1, \qquad \forall i \in \mathcal{P} \quad (5.7)$$

$$x_{is}^{tk} = 0 \qquad \forall i \in P, k \in K, s \in S, t \in \{0, \ldots, r_i\} \quad (5.8)$$

$$x_{is}^{tk} \le 1 - x_{i's'}^{tk}, \qquad \forall i, i' \in \mathcal{P}, k \in \mathcal{K}, t \in \mathcal{T}$$

$$s = 0, \ldots, |\mathcal{S}| - p_i + 1,$$

$$s' = s+1, \ldots, s + p_i - 1 \quad (5.9)$$

$$x_{is}^{tk} = 0, \qquad \forall t \in \mathcal{T}, i \in \mathcal{P}, k \in \mathcal{K},$$

$$s = |\mathcal{S}| - p_i + 1, \ldots |\mathcal{S}| - 1 \quad (5.10)$$

$$z_i^t \geq \sum_{k\in K}\sum_{s\in S} x_{is}^{tk} - \sum_{k\in K}\sum_{s\in S} x_{is}^{t-1,k}, \qquad \forall i \in P, t \in T, x_{is}^{-1,k} = 0 \tag{5.11}$$

$$w_{ik} \geq x_{is}^{tk} \qquad \forall i \in P, t \in T, k \in K, s \in S \tag{5.12}$$

$$\bar{\Delta}_i^t \geq x_{is}^{tk}(t_i^{min} - s), \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \hat{\mathcal{P}}^N, s \in \mathcal{S} \tag{5.13}$$

$$\underline{\Delta}_i^t \geq x_{is}^{tk}(s - t_i^{max}), \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \hat{\mathcal{P}}^N, s \in \mathcal{S} \tag{5.14}$$

$$\bar{\theta}_i \geq \sum_{s\in\mathcal{S}} s x_{is}^{tk}, \qquad \forall t \in \mathcal{T}, i \in \hat{\mathcal{P}}^N, k \in \mathcal{K} \tag{5.15}$$

$$\underline{\theta}_i \leq \sum_{s\in\mathcal{S}} s x_{is}^{tk}, \qquad \forall t \in \mathcal{T}, i \in \hat{\mathcal{P}}^N, k \in \mathcal{K} \tag{5.16}$$

$$\bar{\phi}_i^t \geq \bar{s}_i^t - \sum_{s\in\mathcal{S}} s x_{is}^{tk}, \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \bar{\mathcal{P}} \tag{5.17}$$

$$\underline{\phi}_i^t \geq \sum_{s\in\mathcal{S}} s x_{is}^{tk} - \bar{s}_i^t, \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \bar{\mathcal{P}} \tag{5.18}$$

Objective function (5.1) minimizes the following objectives: (i) the waiting days of patients; (ii) the number of linacs assigned for each patient; (iii) the deviations from the preference time windows; (iv) the inconsistency in appointment times; and (v) the changes in appointment times of fixed patients. ω_1 to ω_5 are the weights of those objectives respectively. Constraints (5.2) ensure that each patient is scheduled at most one session per day. Constraints (5.3) ensure that at most one patient is scheduled for each slot on each linac per day. Constraints (5.4) prevent assigning patients to non-eligible linacs. Constraints (5.5) make sure each patient is scheduled less than the required number of sessions. Constraints (5.6) take care of the continuity of treatment (daily treatments) until the end of the horizon or until the required number of sessions met. Constraints (5.7) and (5.8) make sure all patients are scheduled before their due day but after their release day. Constraints (5.9) ensure the number of required time slots for each patient by blocking the next slots on the linacs. Constraints (5.10) ensure no patients are scheduled on the last slot(s) of the days which are not sufficient for their

fraction. Constraints (5.11) define the first days of treatment of patients. Constraints (5.12) are linking constraints between w and x. Constraints (5.13) and (5.14) define the deviation from the preference time window. Constraints (5.15) and (5.16) define the earliest and latest appointment time of a patient during his/her treatment course. Constraints (5.17) and (5.18) define the changes in appointment times of fixed patients

5.3 A two-phase approach

In practice, minimizing the waiting time of patients is the most important objective and is considered foremost before any other preferences in many cancer centers. This can be simulated in the above model by setting the weight of the first term of the objective function (ω_1) much larger than the weights of the remaining terms. When ω_1 is large enough, solving the problem is equivalent to solving two sub-problems sequentially: (1) determining the assignments of fractions to days and linacs; (2) given the assignments patient-day-linac, deciding the sequence of patients and the exact appointment times. We hereby propose separate models to solve these two phases sequentially.

5.3.1 Phase 1: assigning patients to days and linacs

Phase 1 decides the starting days of patients and also assigns the treatments to specific linacs. We propose a MIP model for phase 1. This model only works on the set of new patients $\hat{\mathcal{P}}$. The following (binary) variables are introduced.

- y_{ik}^t = 1 if the new patient i is assigned to day t on linac k, 0 otherwise.
- z_i^t = 1 if the new patient i receives his/her first treatment on day t, 0 otherwise.
- w_{ik} = 1 if the new patient i is assigned to linac k during his/her treatment.

Model:

$$\text{minimize } \omega_1 \sum_{i\in\hat{\mathcal{P}}} \sum_{t\in T} t z_i^t + \omega_2 \sum_{i\in\hat{\mathcal{P}}} \sum_{k\in K} w_{ik} \tag{5.19}$$

subject to

$$\sum_{k\in K} y_{ik}^t \leq 1 \qquad \forall i \in \hat{\mathcal{P}}, t \in T \tag{5.20}$$

$$y_{ik}^t = 0, \qquad \forall i \in \hat{\mathcal{P}}, t \in \mathcal{T}, k \notin \mathcal{K}^i \tag{5.21}$$

$$\sum_{k\in K}\sum_{t\in T} y_{ik}^{t} \leq I_i \qquad \forall i \in \hat{\mathcal{P}} \quad (5.22)$$

$$\sum_{k\in K}(y_{ik}^{t} - y_{ik}^{t-1}) \leq \sum_{k\in K} y_{ik}^{n} \qquad \forall i \in \hat{\mathcal{P}}, t \in T,$$

$$n = t+1, \ldots, min\{|T|-1, t+I_i-1\}, y_{ik}^{-1} = 0 \quad (5.23)$$

$$\sum_{k\in K}\sum_{t=0}^{d_i} y_{ik}^{t} \geq 1 \qquad \forall i \in \hat{\mathcal{P}} \quad (5.24)$$

$$y_{ik}^{t} = 0 \qquad \forall i \in \hat{\mathcal{P}}, k \in K, t \in \{0, \ldots, r_i\} \quad (5.25)$$

$$\sum_{i\in\hat{\mathcal{P}}} p_i \times y_{ik}^{t} \leq \hat{C}_k^t \qquad \forall t \in T, k \in K \quad (5.26)$$

$$z_i^t \geq \sum_{k\in K} y_{ik}^{t} - \sum_{k\in K} y_{ik}^{t-1}, \qquad \forall i \in \hat{\mathcal{P}}, t \in T, y_{ik}^{-1} = 0 \quad (5.27)$$

$$w_{ik} \geq y_{ik}^{t} \qquad \forall i \in \hat{\mathcal{P}}, t \in T, k \in K \quad (5.28)$$

Objective function (5.19) minimizes the waiting times and the total number of linacs assigned to a patient during his/her treatment. Constraints (5.20) ensure each patient is scheduled on at most 1 linac per day. Constraints (5.21) prevent assigning patients to non-eligible linacs. Constraints (5.22) limit the number of sessions delivered to each patient. Constraints (5.23) ensure daily treatments. Constraints (5.24) and (5.25) make sure all patients are treated before their due days but after their release days. Constraints (5.26) ensure the capacity of each linac is not exceeded where $\hat{C}_k^t$ is the available capacity of linac k, day t after deducting the treatments of fixed patients. Constraints (5.27) define the first days of treatment. Constraints (5.28) keep track of a patient being assigned to a linac during his/her course of treatment.

5.3.2 Phase 2: ordering patients on linacs - IP model

Given the assignment of patients to days and linacs, let P_k^t be the set of patients scheduled on linac k, day t. P_k^t is composed of two subsets $\bar{P}_k^t$, including fixed patients from the previous period and $\hat{P}_k^t$, including new patients. The linac assigned to patient i on day t is known from the first phase and denoted as k_i^t.

The variables used in the model are similar to the variables used in the first model for the whole problem.

- x_{is}^{tk}: binary variables indicating that patient i is assigned to slot s.
- $\bar{\Delta}_i^t$ and $\underline{\Delta}_i^t$: representing the deviation from the preference time window of non-palliative patient $i \in \mathcal{P}^N$ on day t (sooner or later).
- $\bar{\theta}_i$ and $\underline{\theta}_i$: representing patient's earliest and latest appointment time slot.
- $\bar{\phi}_i^t$ and $\underline{\phi}_i^t$: representing changes in appointment times (to sooner and later) of fixed patient $i \in \bar{\mathcal{P}}$ on day t.

Model:

$$\text{minimize } \omega_3 \sum_{i \in \mathcal{P}^N, t \in T} (\bar{\Delta}_{it} + \underline{\Delta}_{it}) + \omega_4 \sum_{i \in \mathcal{P}^N} (\bar{\theta}_i - \underline{\theta}_i) + \omega_5 \sum_{i \in \bar{\mathcal{P}}, t \in T} (\bar{\phi}_i^t + \underline{\phi}_i^t) \tag{5.29}$$

subject to

$$\sum_{s \in S} x_{is}^{tk} = 1 \qquad \forall t \in T, k \in K, i \in P_k^t \tag{5.30}$$

$$\sum_{i \in P_k^t} x_{is}^{tk} \leq 1 \qquad \forall t \in T, k \in K, s \in S \tag{5.31}$$

$$x_{is}^{tk} \leq 1 - x_{i's'}^{tk}, \qquad \forall t \in T, k \in \mathcal{K}, i, i' \in P_k^t, i \neq i'$$
$$s = 0, \ldots, |\mathcal{S}| - p_i$$
$$s' = s+1, \ldots, s+p_i-1 \tag{5.32}$$

$$x_{is}^{tk} = 0, \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, i \in \mathcal{P}_k^t,$$
$$s = |\mathcal{S}| - p_i + 1, ...|\mathcal{S}| - 1 \tag{5.33}$$

$$\bar{\Delta}_i^t \geq x_{is}^{tk}(t_i^{min} - s), \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \hat{\mathcal{P}}_k^{Nt}, s \in \mathcal{S} \tag{5.34}$$

$$\underline{\Delta}_i^t \geq x_{is}^{tk}(s - t_i^{max}), \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \hat{\mathcal{P}}_k^{Nt}, s \in \mathcal{S} \tag{5.35}$$

$$\bar{\theta}_i \geq \sum_{s \in \mathcal{S}} s x_{is}^{tk}, \qquad \forall t \in \mathcal{T}, i \in \hat{\mathcal{P}}^{Nt}, k = k_i^t \tag{5.36}$$

$$\underline{\theta}_i \leq \sum_{s \in \mathcal{S}} s x_{is}^{tk}, \qquad \forall t \in \mathcal{T}, i \in \hat{\mathcal{P}}^{Nt}, k = k_i^t \tag{5.37}$$

$$\bar{\phi}_i^t \geq \bar{s}_i^t - \sum_{s \in \mathcal{S}} s x_{is}^{tk}, \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \bar{\mathcal{P}}_k^t \tag{5.38}$$

$$\underline{\phi}_i^t \geq \sum_{s \in \mathcal{S}} s x_{is}^{tk} - \bar{s}_i^t, \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \bar{\mathcal{P}}_k^t \tag{5.39}$$

Objective function (5.29) minimizes the deviation from preference time windows, inconsistency in appointment times and changes of appointments of fixed patients with the respective weights ω_3, ω_4 and ω_5. Constraints (5.30) make sure each patient is assigned to exactly one slot in their treatment day. Constraints (5.31) make sure each slot is assigned to at most one patient. Constraints (5.32) ensure the number of required time slots for each patient by blocking the next slots on the linacs. Constraints (5.33) ensure no patients are scheduled on the last slot(s) of the days which are not sufficient for their treatment time. Constraints (5.34) and (5.35) define the deviation from the time preferences. Constraints (5.36) and (5.37) define the earliest and latest appointment time of a patient during his/her treatment course. Constraints (5.38) and (5.39) define the changes in appointment times of fixed patients.

5.3.3 Phase 2: order of patients on linacs - CP model

As an alternative for the MIP model, a Constraint Programming model is proposed for phase 2. The model uses variables and constraints supported by IBM CP Optimizer (Laborie et al., 2018). CP Optimizer supports two main types of decision variables, namely *integer* and *interval variables*. An integer variable is a discrete integer while an interval variable consists of a start point and an end point, representing an interval of time. Another type of variable used in this model is a *sequence variable*. A sequence variable consists of a set of interval variables.

In this model, the parameters are similar to the ones in the MIP models. We hereby introduce the variables and constraints.

- The interval variable y_i^t represents the appointment duration of patient i on day t. The duration (d) of variable y_i^t is defined by the start (s) and the end (e) of the fraction of the corresponding patient. Recall that C_k^t is the capacity of linac k in day t.

$$\forall t \in \mathcal{T}, \forall k \in \mathcal{K}, i \in \mathcal{P}_k^t : \begin{cases} s(y_{ki}^t) \in [0, C_k^t - p_i] \\ e(y_{ki}^t) \in [C_k^t - p_i, C_k^t] \\ d(y_{ki}^t) = p_i \end{cases} \tag{5.40}$$

- The interval variable x_i represents the time span of appointment times of patient i. x_i starts at the earliest appointment time and ends at the latest appointment end time of patient i during his/her treatment period. The longer the duration of x_i,

the more inconsistency appointment times of the corresponding patient during the treatment. This is imposed by the specialized constraint *span* of CPOptimizer.

$$i \in \mathcal{P}_k^t : \begin{cases} s(x_i) \in [0, C_k^t] \\ e(x_i) \in [0, C_k^t] \\ span(x_i, [y_{ki}^t | \forall t \in \mathcal{T}, \forall k \in \mathcal{K}]) \end{cases} \tag{5.41}$$

- The *sequence variable* seq_k^t contains all interval variables y_{ki}^t corresponding to all fractions performed on linac k on day t. The specialized constraint *noOverlap* is imposed on seq_k^t to make sure that a fraction must end before the next one starts.

$$\forall t \in \mathcal{T}, k \in \mathcal{K} : \begin{cases} seq_k^t = \{y_{ki}^t | i \in P_k^t\} \\ \text{noOverlap}(seq_k^t) \end{cases} \tag{5.42}$$

- $\bar{\Delta}_i^t$ and $\underline{\Delta}_i^t$ represent the deviation (to sooner or later) from the preference time windows of new, non-palliative patients. They are defined by constraints (5.44).

$$\forall t \in \mathcal{T}, i \in \hat{\mathcal{P}} : \begin{cases} \bar{\Delta}_i^t \in [0, C_k^t] \\ \underline{\Delta}_i^t \in [0, C_k^t] \end{cases} \tag{5.43}$$

$$\forall t \in \mathcal{T}, k \in \mathcal{K}, i \in \hat{\mathcal{P}} : \begin{cases} \bar{\Delta}_i^t \geq t_i^{min} - StartOf(y_{ki}^t) \\ \underline{\Delta}_i^t \geq StartOf(y_{ki}^t) - t_i^{max} \end{cases} \tag{5.44}$$

The objective function (5.45) consists of three parts, each of which minimizes the following objectives respectively: (1) the violation of time window preference of patients; (2) in-consistency in appointment times; and (3) changes in appointments of fixed patients.

$$\begin{aligned} \text{minimize } & \omega_3 \sum_{t \in \mathcal{T}} \sum_{i \in P_k^t} (\bar{\Delta}_i^t + \underline{\Delta}_i^t) \\ & + \omega_4 \sum_{i \in \mathcal{P}^N} LengthOf(x_i) \\ & + \omega_5 \sum_{i \in \bar{\mathcal{P}}_k^t} abs(StartOf(y_{ki}^t) - \bar{s}_i^t) \end{aligned} \tag{5.45}$$

Category	Percentage	Deadline (days)
P1	0.44%	1
P2	27.14%	3
P3	41.36%	14
P4	31.06%	28

Table 5.1: Distribution of patient categories.

5.4 Experiment

5.4.1 Data description

In this section, we present an overview of the data from CHUM and how we generate test instances based on real data. CHUM is a large cancer center in Montréal, Canada. In 2018, CHUM has around 4400 consultations, 3500 new patients and 40.000 treatments. Currently, the scheduling is made manually. The center is equipped with 11 linacs, divided into 4 groups. Linacs from each group are almost identical and patients can switch between them during treatments. The scheduling is conducted sequentially, i.e. the appointments are made once the treatment plan of a new patient is decided. Usually, the next week is full with some slots for emergency treatments, the second week is about 70% full while the third week has a couple of booked slots. Preferences in appointment times only apply for some patients which live far away from the center. Based on the diagnosis of the patients, the doctors will decide their treatment plans. Currently, there are 356 treatment plans. Each treatment plan is associated with a priority level and one or two groups of eligible linacs. There are 4 types of priority, from P1 to P4. Patients with P1 and P2 priority are palliative ones which require urgent treatment. Patients with P2 and P3 priorities are curative ones which have 14 and 28 days of due day deadline respectively. The percentages of patients regarding categories and the deadlines are listed in table 5.1. The number of sections and fraction lengths depend on both the treatment plan and the specific condition of the patient. However, each treatment plan usually has a small range of fraction length values. All fraction length values are multiples of 5 minutes.

5.4.2 Data generation

We generate test data based on the distribution of real data from CHUM. Data is generated using the following steps:

1. A pool of patients is generated. Each patient has a treatment plan, which also determines his/her priority, number of fractions, fraction length and eligible linac

groups. This information is generated based on the distribution of the real data.

2. An instance size is determines by the number of linacs (can be limited in both the number of linacs in each group and the number of groups), the planning horizon (number of days that the schedule should be made), and the number of new patients.

3. Given an instance, fixed patients are assigned so that the occupation rate of each week in line with real data. To do so, non-palliative patients are iteratively taken from the patient pool and assigned to a series of free slots in the schedule, until the desired occupation rates are met (a series is a set of fractions with identical appointment time in a couple of consecutive days).

4. Finally, a pre-defined number of patients is taken from the patient pool as the new patients for the instance. The release date of each patient is generated randomly. The due day of each patient is then calculated according his/her release date and urgency level. Currently, to simplify the problem, we generate release dates of patients so that their due days are always within the planning horizon.

5.4.3 Numerical results

All experiments are carried out on an Intel Core i7-4600 2.90GHz * 4 machine with 8GB of RAM, running Ubuntu 16.04 64-bit. IBM ILOG CPLEX and CP Optimizer version 12.8.0 are used as the solvers.

The test set consists of 16 instances with a horizon of 21 days. The numbers of linacs range from 1 to 7, divided into up to 3 groups. The numbers of patients range from 10 to 80 patients and are adjusted according to the number of linacs so that the instances are feasible. Phase 1 is given 10 seconds of time budget while in phase 2, both MIP and CP are given 30 seconds of time limit. The results can be found in table 5.2. The first column shows instance names, the next three columns display number of linacs of each group, the fifth column is number of patients. For each model, the objective values, optimality gaps, solving times and best bounds are reported.

It can be seen from the results that phase 1 is easy to solve. Optimal solutions are achieved in all instances within less than 8 seconds, with the exception of 95 seconds in instance *ins15*. In phase 2, CP tremendously outperforms MILP in all instances. Given only 30 seconds of run time, CP Optimizer achieves optimal solutions in 9 out of 16 instances. In the remaining 7 instances, good objective values are also obtained with 12% to 81% gap from the lower bounds. By contrast, CPLEX does not perform very well in most instances, except *ins01* and *ins03* where optimal solutions are obtained. The remaining instances in general have bad objective values compared to results from CP. Especially, the lower bounds offered by CPLEX are either negative or 0, which

					Phase 1			Phase 2 MIP				Phase 2 CP			
Ins	K1	K2	K3	P	obj	gap (%)	time	obj	time	gap (%)	bound	obj	time	gap (%)	bound
ins01	1	0	0	10	31	0	0.06	**0**	6.34	0	0	**0**	1.17	0	0
ins02	1	0	0	25	35	0	0.05	7665	30.26	128	-2164	**25**	30.13	12	8
ins03	2	0	0	20	23	0	0.23	**0**	9.82	0	0	**0**	2.02	0	0
ins04	2	0	0	30	36	0	0.27	10714	30.20	126	-2743	**0**	16.19	0	0
ins05	2	0	0	40	47	0	0.49	12325	30.26	132	-3891	**0**	3.95	0	0
ins06	2	1	0	40	3571	0	0.64	12861	30.27	121	-2720	**0**	5.76	0	0
ins07	2	1	0	50	55	0	0.63	12724	30.37	133	-4304	**69**	30.23	19	0
ins08	2	1	0	60	271	0	0.62	12752	30.39	133	-4266	**650**	30.30	81	2
ins09	3	0	0	40	40	0	1.92	459	30.28	100	0	**0**	17.93	0	0
ins10	3	0	0	50	58	0	2.53	17946	30.25	126	-4692	**0**	28.10	0	0
ins11	3	0	0	60	60	0	8.15	13077	30.45	146	-5955	**429**	30.27	63	0
ins12	3	1	1	50	110	0	0.87	24	30.08	100	0.0	**0**	15.32	0	0
ins13	3	1	1	60	60	0	1.67	16881	30.66	128	-4691	**186**	30.25	35	0
ins14	3	1	1	70	131	0	1.46	25153	30.48	122	-5454	**49**	30.37	14	0
ins15	4	2	1	70	3473	0	86.90	21126	30.67	123	-4764	**327**	30.37	59	0
ins16	4	2	1	80	138	0	3.26	21738	30.83	126	-5669	**0**	18.02	0	0

Table 5.2: Results with 30 seconds time limit.

					Phase 2 MIP				Phase 2 CP			
Ins	K1	K2	K3	P	obj	time	gap (%)	bound	obj	time	gap (%)	bound
ins01	1	0	0	10	**0**	5.89	0	0.0	**0**	0.99	0	0
ins02	1	0	0	25	441	1200.11	100	1.6	**19**	233.04	0	19
ins03	2	0	0	20	**0**	9.47	0	0.0	**0**	2.02	0	0
ins04	2	0	0	30	**0**	183.76	0	0.0	**0**	16.10	0	0
ins05	2	0	0	40	344	1200.38	100	0.0	**0**	3.97	0	0
ins06	2	1	0	40	**0**	37.72	0	0.0	**0**	5.77	0	0
ins07	2	1	0	50	973	1200.15	100	0.3	**3**	286.15	0	3
ins08	2	1	0	60	743	1200.13	100	0.2	**0**	54.73	0	4
ins09	3	0	0	40	**0**	210.25	0	0.0	**0**	18.07	0	0
ins10	3	0	0	50	42	1200.14	100	0.0	**0**	28.36	0	0
ins11	3	0	0	60	295	1200.42	100	0.0	**0**	49.79	0	0
ins12	3	1	1	50	**0**	52.44	0	0.0	**0**	15.24	0	0
ins13	3	1	1	60	7	1200.33	100	0.0	**0**	45.26	0	0
ins14	3	1	1	70	**0**	313.13	0	0.0	**0**	35.94	0	0
ins15	4	2	1	70	**29**	802.17	0	29.0	32	1200.30	10	3
ins16	4	2	1	80	**0**	280.61	0	0.0	**0**	18.12	0	0

Table 5.3: Results with 1200 seconds time limit.

results in large optimality gaps. This can be explained by the appointment consistency terms in the objective function of the MILP model.

In table 5.3, the results with a larger time limit (1200 seconds) are reported. With the large time limit, CP Optimizer achieves optimal solutions in all instances while CPLEX closes the gaps in 9 out of 16 instances. In the remaining 7 instances, CPLEX also obtains good solutions compared to the results with 30 seconds, but the optimality gaps are still large. We come to the conclusion that CP performs very well in the RTSP while MILP might benefit from another formulation which can offer better bounds.

5.5 Conclusions and future work

In this chapter, we have introduced a 2-phase approach for the Radiotherapy Scheduling Problem. A MIP model is proposed for phase 1 while a MIP and a CP model are proposed for phase 2. We also present how realistic instances are generated based on real instances from a cancer center in Montréal. The numerical results show that CP outperforms MIP in all instances and is able to solve realistic instance sizes.

As for future work, we aim to solve large-sized instances with more sophisticated algorithms. It is also interesting to take into account cancellations and future changes in the schedule.

Chapter 6

Conclusion

In this chapter, we summarize the content of this dissertation and highlight the results as well as our contributions to literature. Moreover, future research directions are discussed.

6.1 Summary and contributions

The ubiquitous essence of combinatorial optimization problems leads to high diversity in solving paradigms, each of which has its own characteristics. These characteristics are potentially complementary to each other, hence investigating the possibility to combine different approaches might be beneficial for solving combinatorial problems. This dissertation exploits this research theme at two different levels.

The first part, composed of chapter 2 and 3, exploits the relationships between solving paradigms at a general level. We investigate the combination of two subdomains of Operations Research and Artificial Intelligence, namely Local Search and Knowledge Representation. In chapter 2, we discuss the relationship between the two domains, analyse their position in the solving of combinatorial problems, through which we identify the links between them. These analyses serve as the motivation for the "declarative local search" presented in chapter 3. Declarative local search is a framework that brings local search and knowledge representation together under the same roof and combine their complementary characteristics. It is built on top of IDP (De Cat et al., 2018), a Knowledge Base System which provides various inference methods to solve different tasks around a center knowledge base described in its formal input language FO(·), an extension of first-order logic. IDP's references for combinatorial

problems, namely *model expansion* and *optimization*, rely on MINISAT(ID) (De Cat et al., 2013b), an exact Constraint Programming-SAT-based solver. Declarative local search allows for declaratively specifying local search-based heuristics using FO(·). The local search specifications are then interpreted by IDP to simulate the algorithms. It is introduced as a new inference for solving combinatorial optimization problems in IDP. The usage of the framework is demonstrated by modelling a set of neighbourhood moves for the classical Travelling Salesman Problem. 15 algorithms are synthesized from 5 neighbourhood moves, which shows the potential of using the framework as a fast prototype tool for local search algorithms. The synthesized algorithms outperform the optimization inference of IDP and the ASP solver *clingo*. This proves the framework's ability to serve as an efficient alternative to solve optimization problems that are conventionally difficult for traditional approaches of IDP in particular and of formal systems in general. In conclusion, for the first part, the declarative local search framework that we propose has made meaningful contributions to both local search and knowledge representation by combining them into one single framework.

In contrast to the first part, the second part focuses on the combination of solving approaches at a low level. We investigate two particular case studies, each of which is inspired by or based on a real-world problem. Different solving methods are proposed for each problem and the comparison/relationships between them are investigated.

Chapter 4 presents The Intermittent Travelling Salesman Problem (ITSP) which is inspired by real-world drilling/texturing applications where the temperature of the work-piece is taken into account. Due to the heat generated during the procedure, the processing of each node has to be divided into several visits with a time lapse in between to let the work-piece cool down. The problem has not been studied before in literature. We model the ITSP as a variant of the Travelling Salesman Problem and propose a decomposition model which is integrated into an exact branch-and-bound approach. We furthermore develop four Variable Neighbourhood Search metaheuristics which produce high quality solutions for large instances. An instance library is created and made publicly available for future research.

The second case study, presented in chapter 5, is the Radiotherapy Scheduling Problem (RTSP). Radiotherapy treatment is a popular form of cancer treatment where a patient receives daily doses of radiation delivered by a linear accelerator (linac). The RTSP in this work is a real problem at CHUM, a large cancer center in Montréal, Canada, where the scheduling is currently made manually. The increasing number of patients causes lots of distress on the scheduling team and lengthens the waiting time of patients, which has a negative impact on the treatment outcomes. We propose a two-phase approach which assigns patients to linacs and days in the first phase and sequences patients on each linac in the second phase. A Mixed Integer Linear Programming (MIP) model is proposed for the first phase while a MIP and a Constraint Programming (CP) model are proposed for the second phase. The algorithm is tested on the dataset generated based on real data from CHUM. We are able to solve close-to-realistic instances in

terms of instance sizes and characteristics, which opens up an opportunity to apply the algorithm in the real context at CHUM. The numerical results show that the CP approach outperforms MIP for this problem, which demonstrates the potential of a non-conventional approach in radiotherapy scheduling and also suggests the possibility of hybrid approaches. This work hence helps to narrow the gaps between two solving paradigms in the problem. Finally, to the best of our knowledge, we are the first to describe how realistic, i.e. partially filled instances are generated based on real data. Since the schedule's layout and characteristics deeply affect the algorithm's performance, this is a meaningful contribution towards closing the gaps between theoretical research and practical problems.

Both case studies in the second part contribute new knowledge to the respective literature. Different solving approaches, both exact and heuristic, are proposed and compared. New data sets are introduced and made publicly available for future research. The two parts of the dissertation together have brought different solving paradigms closer to each other and towards the ultimate goal of solving combinatorial optimization problems efficiently and seamlessly.

6.2 Future research directions

There are many potential research directions for this dissertation. Regarding declarative local search, extending the framework to allow for more flexibility in metaheuristics modelling, e.g. stopping criteria and adaptive selection of neighbourhood moves, is a promising direction to improve the modelling power of the framework. Another direction is to exploit the facilities provided by IDP. Formal languages, backed by more than 40 years of research, offer many inference methods, which might be utilized in local search algorithms. For example, we anticipate that *automated theorem proving* can be used to verify if a neighbourhood move is correct. Another possibility is to extract knowledge from the formal specifications of a problem to generate algorithms automatically. Preliminary work has been done by Devriendt et al. (2015) where symmetry information is exacted from the problem specifications to generate neighbourhood moves automatically.

In the second part, each case study has lots of potentials to expand. The ITSP is a new and challenging problem that is open for further improvement, especially in terms of exact/hybrid approaches. The RTSP is a meaningful problem that has big impact on the treatment outcomes of cancer patients. In the near future, we have two goals. Firstly, we want to extend the problem with more constraints so that it becomes flexible and can be applied to many centers. Second, we want to take into account uncertainty factors, e.g. the patients have to delay their treatments or change their treatment plans, which is very popular in healthcare.

As a closing remark, we recall the goal of this dissertation on narrowing the gaps between different solving paradigms. With declarative local search, we have made an attempt to unify two solving approaches from different fields into one framework. Encouraging results were achieved, showing the potential of the approach. Local search heuristics benefits from the declarative power of the adopted formal language while the solving ability of the logic solver IDP is extended in some classes of problems with the local search option. The two case studies in part II are a demonstration of how solving a problem should not be bound to a specific method. A problem can be decomposed and different solving approaches should be investigated to solve the sub-problems based on their combinatorial structures, characteristics or the intentions of the operations research practitioner. If an optimal answer is a must, exact methods should be applied. If a fast answer is required, heuristic methods should be considered first. If a flexible model is preferable, logic or constraint programming should be investigated. From this observation one may argue that different solving methods should all be accessible from one system or at least, the hassle of extra work to combine them should be minimized. This dissertation highlights this need and we believe that declarative local search is a firm step towards this direction. There are still limitations to improve upon. There are open questions concerning efficiency or concerning expression of incompleteness for heuristics and how this can match the guarantees present in knowledge based or logic based systems. This dissertation, of course, is not the first one to investigate this topic and definitely will not be the last. Continued research is hence indispensable to achieve this goal.

Bibliography

Canadian Cancer Statistics, a 2018 special report. http://www.cancer.ca/en/cancer-information/cancer-101/canadian-cancer-statistics-publication/?region=on, 2018. [Online; accessed 08-August-2019].

María J. Blesa Aguilera, Christian Blum, Haroldo Gambini Santos, Pedro Pinacho Davidson, and Julio Godoy del Campo, editors. *Hybrid Metaheuristics - 11th International Workshop, HM 2019, Concepción, Chile, January 16-18, 2019, Proceedings*, volume 11299 of *Lecture Notes in Computer Science*, 2019. Springer. ISBN 978-3-030-05982-8. doi: 10.1007/978-3-030-05983-5. URL https://doi.org/10.1007/978-3-030-05983-5.

Ozgur Akgun, Saad Wasim A Attieh, Ian Philip Gent, Christopher Anthony Jefferson, Ian James Miguel, Peter William Nightingale, András Z Salamon, Patrick Spracklen, and James Patrick Wetter. A framework for constraint based local search using Essence. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 1242–1248. International Joint Conferences on Artificial Intelligence, 2018.

Mario Alviano, Carmine Dodaro, Nicola Leone, and Francesco Ricca. Advances in WASP. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 40–54. Springer, 2015.

Mario Alviano, Carmine Dodaro, and Marco Maratea. An advanced answer set programming encoding for nurse scheduling. In *Conference of the Italian Association for Artificial Intelligence*, pages 468–482. Springer, 2017.

David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde TSP solver, 2006.

Ionuţ Aron, John N Hooker, and Tallys H Yunes. Simpl: A system for integrating optimization techniques. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 21–36. Springer, 2004.

Marcello Balduccini and Michael Gelfond. Diagnostic reasoning with A-Prolog. *TPLP*, 3(4–5):425–461, 2003.

Marcello Balduccini, Yuliya Lierler, and Peter Schüller. Prolog and ASP inference under one roof. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 148–160. Springer, 2013.

Chitta Baral and Cenk Uyan. Declarative specification and solution of combinatorial auctions using logic programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 186–199. Springer, 2001.

Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors. *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, volume 4483 of *Lecture Notes in Computer Science*, 2007. Springer. ISBN 978-3-540-72199-4.

Thierry Benoist, Bertrand Estellon, Frédéric Gardi, Romain Megel, and Karim Nouioua. Localsolver 1. x: a black-box local-search solver for 0-1 programming. *4OR: A Quarterly Journal of Operations Research*, 9(3):299–316, 2011.

Gustav Björdal, Jean-Noël Monette, Pierre Flener, and Justin Pearson. A constraint-based local search backend for MiniZinc. *Constraints*, 20(3):325–345, 2015.

Gustav Björdal, Pierre Flener, Justin Pearson, Peter J Stuckey, and Guido Tack. Declarative Local-Search Neighbourhoods in MiniZinc. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 98–105. IEEE, 2018.

F Bock. An algorithm for solving “traveling salesman" and related network optimization problems, presented at the 14th national meeting of the operations research society of america, st. *Louis, Missouri*, 1958.

Kenneth D Boese, Andrew B Kahng, and Sudhakar Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16 (2):101–113, 1994.

Maurice Bruynooghe, Hendrik Blockeel, Bart Bogaerts, Broes De Cat, Stef De Pooter, Joachim Jansen, Anthony Labarre, Jan Ramon, Marc Denecker, and Sicco Verwer. Predicate logic as a modeling language: modeling and solving some machine learning and data mining problems with IDP3. *Theory and Practice of Logic Programming*, 15(6):783–817, 2015.

Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of metaheuristics*, pages 457–474. Springer, 2003.

Edmund K Burke and Yuri Bykov. The late acceptance hill-climbing heuristic. *Computing Science and Mathematics, University of Stirling, Tech. Rep. Technical Report No. CSM-192*, 2012.

Edmund K Burke, Peter I Cowling, and Ralf Keuthen. Effective local and guided variable neighbourhood search methods for the asymmetric travelling salesman problem. In *Workshops on Applications of Evolutionary Computation*, pages 203–212. Springer, 2001.

Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

Ann Campbell, Lloyd Clarke, Anton Kleywegt, and Martin Savelsbergh. The inventory routing problem. In *Fleet management and logistics*, pages 95–113. Springer, 1998.

Zheng Chen, Will King, Robert Pearcey, Marc Kerba, and William J Mackillop. The relationship between waiting time for radiotherapy and clinical outcomes: a systematic review of the literature. *Radiotherapy and Oncology*, 87(1):3–16, 2008.

CE Coles, L Burgess, and LT Tan. An audit of delays before and during radical radiotherapy for cervical cancer–effect on tumour cure probability. *Clinical oncology*, 15(2):47–54, 2003.

Alain Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7): 69–90, 1990. ISSN 0001-0782. doi: http://dx.doi.org/10.1145/79204.79210.

Domenico Conforti, Francesca Guerriero, and Rosita Guido. Optimization models for radiotherapy patient scheduling. *4OR*, 6(3):263–278, 2008.

Domenico Conforti, Francesca Guerriero, and Rosita Guido. Non-block scheduling with priority for radiotherapy treatments. *European Journal of Operational Research*, 201(1):289–296, 2010.

Marcos Cramer, Pieter Van Hertum, Diego Agustin Ambrossio, and Marc Denecker. Modelling Delegation and Revocation Schemes in IDP. *arXiv preprint arXiv:1405.1584*, 2014.

Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

George B Dantzig. Origins of the simplex method. Technical report, Stanford Univ CA Systems Optimization Lab, 1987.

Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

Broes De Cat, Bart Bogaerts, Jo Devriendt, and Marc Denecker. Model expansion in the presence of function symbols using constraint programming. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013*, pages 1068–1075. IEEE Computer Society, 2013a. ISBN 978-1-4799-2971-9. doi: 10.1109/ICTAI.2013.159. URL http://dx.doi.org/10.1109/ICTAI.2013.159.

Broes De Cat, Bart Bogaerts, Jo Devriendt, and Marc Denecker. Model expansion in the presence of function symbols using constraint programming. In *25th International Conference on Tools with Artificial Intelligence, USA, November 4-6, 2013*, pages 1068–1075, 2013b.

Broes De Cat, Bart Bogaerts, Maurice Bruynooghe, Gerda Janssens, and Marc Denecker. Predicate logic as a modeling language: the IDP system. In *Declarative Logic Programming*, pages 279–323. Association for Computing Machinery and Morgan & Claypool, 2018.

Maria Garcia de la Banda, Peter J Stuckey, Pascal Van Hentenryck, and Mark Wallace. The future of optimization technology. *Constraints*, 19(2):126–138, 2014.

Renaud De Landtsheer and Christophe Ponsard. OscaR.cbls: an open source framework for constraint-based local search. *Proceedings of ORBEL*, 27, 2013.

Renaud De Landtsheer, Yoann Guyot, Gustavo Ospina, and Christophe Ponsard. Combining neighborhoods into local search strategies. In *Recent Developments in Metaheuristics*, pages 43–57. Springer, 2018.

Jip J Dekker, Maria Garcia de la Banda, Andreas Schutt, Peter J Stuckey, and Guido Tack. Solver-independent large neighbourhood search. In *International Conference on Principles and Practice of Constraint Programming*, pages 81–98. Springer, 2018.

Jo Devriendt. *Exploiting Symmetry in Model Expansion for Predicate and Propositional Logic*. PhD thesis, Informatics Section, Department of Computer Science, Faculty of Engineering Science, February 2017. URL https://lirias.kuleuven.be/handle/123456789/564687. Denecker, Marc (supervisor).

Jo Devriendt, Patrick De Causmaecker, and Marc Denecker. Transforming constraint programs to input for local search. In *The Fourteenth International Workshop on Constraint Modelling and Reformulation*, pages 1–16, 2015.

Luca Di Gaspero and Andrea Schaerf. EasyLocal++: an object-oriented framework for the flexible design of local-search algorithms. *Software: Practice and Experience*, 33(8):733–765, 2003.

Carmine Dodaro, Philip Gasteiger, Nicola Leone, Benjamin Musitsch, Francesco Ricca, and Kostyantyn Shchekotykhin. Combining answer set programming and domain

heuristics for solving hard industrial problems (application paper). *Theory and Practice of Logic Programming*, 16(5-6):653–669, 2016.

M Drigo. The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1):1–13, 1996.

Yvan Dumas, Jacques Desrosiers, Eric Gelinas, and Marius M Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations research*, 43(2):367–371, 1995.

Russell Eberhart and James Kennedy. Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks*, volume 4, pages 1942–1948. Citeseer, 1995.

Lei Fang and Michael S Hsiao. A new hybrid solution to boost SAT solver performance. In *Proceedings of the conference on Design, automation and test in Europe*, pages 1307–1313. EDA Consortium, 2007.

Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

Andreas Fink and Stefan Voß. HotFrame: a heuristic optimization framework. In *Optimization software class libraries*, pages 81–154. Springer, 2003.

Lawrence J Fogel. Evolutionary programming in perspective: The top-down view. *Computational intelligence: Imitating life*, 1994.

Daniel Fontaine and Laurent Michel. A high level language for solver independent model manipulation and generation of hybrid solvers. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 180–194. Springer, 2012.

Alan M Frisch, Matthew Grum, Christopher Jefferson, Bernadette Martínez Hernández, and Ian Miguel. The design of ESSENCE: A constraint language for specifying combinatorial problems. In *IJCAI*, volume 7, pages 80–87, 2007.

Alan M. Frisch, Warwick Harvey, Christopher Jefferson, Bernadette Martínez Hernández, and Ian Miguel. ESSENCE: a constraint language for specifying combinatorial problems. *Constraints*, 13(3):268–306, 2008.

Laurent Garcia, Claire Lefèvre, Igor Stéphan, Odile Papini, and Eric Würbel. A semantic characterization asp base revision. *Journal of Artificial Intelligence Research*, 66:989–1029, 2019.

Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. *clasp*: A conflict-driven answer set solver. In Baral et al. (2007), pages 260–265. ISBN 978-3-540-72199-4.

Martin Gebser, Torsten Schaub, and Sven Thiele. GrinGo: A new grounder for Answer Set Programming. In Baral et al. (2007), pages 266–271. ISBN 978-3-540-72199-4.

Martin Gebser, Benjamin Kaufmann, Javier Romero, Ramón Otero, Torsten Schaub, and Philipp Wanko. Domain-specific heuristics in answer set programming. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo= ASP+ control: Preliminary report. *arXiv preprint arXiv:1405.3694*, 2014.

Gecode Team. Gecode: Generic constraint development environment, 2013. Available from `http://www.gecode.org`.

Fred Glover and Manuel Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.

Fred Glover, Manuel Laguna, and Rafael Martí. Fundamentals of scatter search and path relinking. *Control and cybernetics*, 29(3):653–684, 2000.

Carla P Gomes. Artificial intelligence and operations research: challenges and opportunities in planning and scheduling. *The Knowledge Engineering Review*, 15(1):1–10, 2000.

Filipe Gouveia, Inês Lynce, and Pedro T Monteiro. Model revision of logical regulatory networks using logic-based tools. In *Technical Communications of the 34th International Conference on Logic Programming (ICLP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

Kannan Govindarajan, Bharat Jayaraman, and Surya Mantha. *Preference Logic Programming: Optimization as Inference*. Department of Computer Science, State University of New York at Buffalo, 1994.

Jun Gu. Efficient local search for very large-scale satisfiability problems. *ACM SIGART Bulletin*, 3(1):8–12, 1992.

Hai-Feng Guo and Bharat Jayaraman. Logic programming with solution preferences. *The Journal of Logic and Algebraic Programming*, 78(1):1–21, 2008.

I Gurobi Optimization. Gurobi optimizer reference manual; 2015. *URL http://www. gurobi. com*, 2015.

Gregory Gutin and Abraham P Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

Djamal Habet, Chu Min Li, Laure Devendeville, and Michel Vasquez. A hybrid approach for sat. In *International Conference on Principles and Practice of Constraint Programming*, pages 172–184. Springer, 2002.

Pierre Hansen and Nenad Mladenović. Variable neighborhood search for the p-median. *Location Science*, 5(4):207–226, 1997.

Pierre Hansen and Nenad Mladenović. Developments of variable neighborhood search. In *Essays and surveys in metaheuristics*, pages 415–439. Springer, 2002.

Pierre Hansen, Nenad Mladenović, and José A Moreno Pérez. Variable neighbourhood search: methods and applications. *4OR*, 6(4):319–360, 2008.

Pierre Hansen, Nenad Mladenović, and José A Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

JN Hooker. Hybrid modeling. Milano M, Van Hentenryck P, eds. Hybrid Optimization—The Ten Years of CPAIOR, Vol. 45, 2011.

John N Hooker. Good and bad futures for constraint programming (and operations research). *Constraint Programming Letters*, 1:21, 2007.

Holger H Hoos and Thomas Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.

Joachim Jansen. Model revision inference for extensions of first order logic. *arXiv preprint arXiv:1405.4206*, 2014.

David S Johnson and Lyle A McGeoch. The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, 1:215–310, 1997.

LV Kantorovich. Mathematical Methods for Production Organization and Planning. *Leningr. Gos. Univ., Leningrad*, 1939.

Truword Kapamara and Dobrila Petrovic. A heuristics and steepest hill climbing method to scheduling radiotherapy patients. In *Proceedings of the 35th International Conference on Operational Research Applied to Health Services, Catholic University of Leuven, Belgium*. Citeseer, 2009.

Ashiqur R KhudaBukhsh, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Satenstein: Automatically building local search sat solvers from components. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

Scott Kirkpatrick, D. Gelatt Jr., and Mario P Vecchi. Optimization by simmulated annealing. *science*, 220(4598):671–680, 1983.

Laura Koponen, Emilia Oikarinen, Tomi Janhunen, and Laura Säilä. Optimizing phylogenetic supertrees using answer set programming. *Theory and Practice of Logic Programming*, 15(4-5):604–619, 2015.

Robert A. Kowalski. Predicate logic as programming language. In *IFIP Congress*, pages 569–574, 1974.

John R Koza and John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250, 2018.

Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

Tiep Le, Tran Cao Son, Enrico Pontelli, and William Yeoh. Solving distributed constraint optimization problems using logic programming. *Theory and Practice of Logic Programming*, 17(4):634–683, 2017.

Antoine Legrain, Marie-Andrée Fortin, Nadia Lahrichi, and Louis-Martin Rousseau. Online stochastic optimization of radiotherapy patient scheduling. *Health care management science*, 18(2):110–123, 2015.

Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

Lengning Liu and Mirosław Truszczyński. WSAT (CC)-a fast local-search ASP solver. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 351–355. Springer, 2004.

Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

Bart L Maccarthy and Jiyin Liu. Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *The International Journal of Production Research*, 31(1):59–79, 1993.

William J. Mackillop. Killing time: the consequences of delays in radiotherapy. *Radiotherapy and Oncology 84.1*, 1(4), 2007.

King-Tim Mak and Andrew J Morton. A modified Lin-Kernighan Traveling-salesman heuristic. *Operations Research Letters*, 13(3):127–132, 1993.

Vittorio Maniezzo and Thomas Stützle, editors. *Matheuristics 2016 - Proceedings of the Sixth International Workshop on Model-based Metaheuristics*, 2016. IRIDIA, Institut de Recherches Interdisciplinaireset de Developpements en Intelligence Artificiell. URL http://iridia.ulb.ac.be/IridiaTrSeries/link/IridiaTr2016-007.pdf.

João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

Kim Marriott, Nicholas Nethercote, Reza Rafeh, Peter J. Stuckey, Maria Garcia de la Banda, and Mark Wallace. The design of the Zinc modelling language. *Constraints*, 13(3):229–267, 2008.

Johannes Maschler and Günther R Raidl. Particle therapy patient scheduling with limited starting time variations of daily treatments. *International Transactions in Operational Research*, 2018.

Johannes Maschler, Martin Riedler, Markus Stock, and Günther R Raidl. Particle therapy patient scheduling: First heuristic approaches. In *Proceedings of the 11th Int. Conference on the Practice and Theory of Automated Timetabling. Udine, Italy*, 2016.

Bertrand Mazure, Lakhdar Saïs, and Éric Grégoire. Boosting complete techniques thanks to local search methods. *Annals of mathematics and artificial intelligence*, 22 (3-4):319–331, 1998.

David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *AAAI/IAAI*, pages 321–326. Rhode Island, USA, 1997.

Xianghu Meng, Jun Li, Xianzhong Dai, and Jianping Dou. Variable Neighborhood Search for a Colored Traveling Salesman Problem. *IEEE Transactions on Intelligent Transportation Systems*, 19(4):1018–1026, 2018.

Laurent Michel and Pascal Van Hentenryck. Localizer: A modeling language for local search. *INFORMS Journal on Computing*, 11(1):1–14, 1999.

Laurent Michel and Pascal Van Hentenryck. The Comet programming language and system. In Peter van Beek, editor, *CP*, volume 3709 of *LNCS*, pages 881–881. Springer, 2005. ISBN 3-540-29238-1.

Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

MA Hakim Newton, Duc Nghia Pham, Abdul Sattar, and Michael Maher. Kangaroo: An efficient constraint-based local search system using lazy propagation. In *International Conference on Principles and Practice of Constraint Programming*, pages 645–659. Springer, 2011.

Ilkka Niemelä, Patrik Simons, and Tommi Syrjänen. Smodels: A system for answer set programming. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*, Breckenridge, Colorado, USA, 2000. CoRR, cs.AI/0003033.

Christian Nilsson. Heuristics for the traveling salesman problem. Technical report, Tech. Report, Linköping University, Sweden, 2003.

Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An A-Prolog decision support system for the space shuttle. In I. V. Ramakrishnan, editor, *PADL*, volume 1990 of *LNCS*, pages 169–183. Springer, 2001. ISBN 3-540-41768-0.

OscaR Team. OscaR: Scala in OR, 2012. Available from `https://bitbucket.org/oscarlib/oscar`.

Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008.

Laurent Perron, Paul Shaw, and Vincent Furnon. Propagation guided large neighborhood search. In *International Conference on Principles and Practice of Constraint Programming*, pages 468–481. Springer, 2004.

Sanja Petrovic and Pedro Leite-Rocha. Constructive approaches to radiotherapy scheduling. In *Proceedings of the World Congress on Engineering and Computer Science*, pages 722–727, 2008.

Sanja Petrovic, William Leung, Xueyan Song, and Santhanam Sundar. Algorithms for radiotherapy treatment booking. In *25th Workshop of the UK planning and scheduling special interest group*, pages 105–112, 2006.

Tu-San Pham and Patrick De Causmaecker. Variable Neighbourhood Search for the Intermittent Traveling Salesman Problem. In *Proceedings of META'2016*, volume 6, pages 57–65, 2016.

Tu-San Pham, Jo Devriendt, and Patrick De Causmaecker. Modelling Local Search in a Knowledge Base System. In *New Trends in Emerging Complex Real Life Problems*, pages 415–423. Springer, 2018a.

Tu-San Pham, Pieter Leyman, and Patrick De Causmaecker. The intermittent travelling salesman problem. *International Transactions in Operational Research*, pages 525–548, 2018b.

Tu-San Pham, Jo Devriendt, and Patrick De Causmaecker. Declarative local search for predicate logic. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 340–346. Springer, 2019a.

Tu-San Pham, Louis-Martin Rousseau, and Patrick De Causmaecker. The Radiotherapy Scheduling Problem. In *Proceedings of MISTA'2019*, 2019b.

David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010.

Petrică C Pop, Levente Fuksz, and Andrei Horvat Marc. A variable neighborhood search approach for solving the generalized vehicle routing problem. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 13–24. Springer, 2014.

Gerhard Reinelt. TSPLIB-A traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

Andrea Rendl, Tias Guns, Peter J Stuckey, and Guido Tack. MiniSearch: a solver-independent meta-search language for MiniZinc. In *International Conference on Principles and Practice of Constraint Programming*, pages 376–392. Springer, 2015.

Francesco Ricca, Giovanni Grasso, Mario Alviano, Marco Manna, Vincenzino Lio, Salvatore Iiritano, and Nicola Leone. Team-building with answer set programming in the gioia-tauro seaport. *Theory and Practice of Logic Programming*, 12(3):361–381, 2012.

Antoine Saure, Jonathan Patrick, Scott Tyldesley, and Martin L Puterman. Dynamic multi-appointment patient scheduling for radiation therapy. *European Journal of Operational Research*, 223(2):573–584, 2012.

Antoine Sauré, Mehmet A Begen, and Jonathan Patrick. Dynamic multi-priority, multi-class patient scheduling with stochastic service times. *European Journal of Operational Research*, 280(1):254–265, 2020.

Tom Schrijvers, Guido Tack, Pieter Wuille, Horst Samulowitz, and Peter J. Stuckey. Search combinators. In Jimmy H.M. Lee, editor, *CP'11*, volume 6876 of *LNCS*. Springer, 2011.

Gerhard Schrimpf, Johannes Schneider, Hermann Stamm-Wilbrandt, and Gunter Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171, 2000.

Bart Selman, Hector J Levesque, David G Mitchell, et al. A new method for solving hard satisfiability problems. In *Aaai*, volume 92, pages 440–446. Citeseer, 1992.

Bart Selman, Henry A Kautz, and Bram Cohen. Noise strategies for improving local search. In *AAAI*, volume 94, pages 337–343, 1994.

Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer, 1998.

Helmut Simonis and Mehmet Dincbas. Using logic programming for fault diagnosis in digital circuits. In *GWAI-87 11th German Workshop on Artifical Intelligence*, pages 139–148. Springer, 1987.

Marques M Sousa, Luiz Satoru Ochi, Igor Machado Coelho, and Luciana Brugiolo Gonçalves. A variable neighborhood search heuristic for the traveling salesman problem with hotel selection. In *Computing Conference (CLEI), 2015 Latin American*, pages 1–12. IEEE, 2015.

Jerry Swan, Steven Adriaensen, Mohamed Bishr, Edmund K Burke, John A Clark, Patrick De Causmaecker, Juanjo Durillo, Kevin Hammond, Emma Hart, Colin G Johnson, et al. A research agenda for metaheuristic standardization. In *Proceedings of the XI Metaheuristics International Conference*, 2015.

Marc Toussaint. Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning. In *IJCAI*, pages 1930–1936, 2015.

Pascal Van Hentenryck and Laurent Michel. Synthesis of constraint-based local search algorithms from high-level models. In *AAAI*, volume 7, pages 273–278, 2007.

Pascal Van Hentenryck and Laurent Michel. The objective-cp optimization system. In *International Conference on Principles and Practice of Constraint Programming*, pages 8–29. Springer, 2013.

Pascal Van Hentenryck, Laurent Michel, Laurent Perron, and J-C Régin. Constraint programming in OPL. In *International Conference on Principles and Practice of Declarative Programming*, pages 98–116. Springer, 1999.

Pieter Van Hertum, Ingmar Dasseville, Gerda Janssens, and Marc Denecker. The KB paradigm and its application to interactive configuration. *Theory and Practice of Logic Programming*, 17(1):91–117, 2017.

Bruno Vieira, Derya Demirtas, Jeroen B van de Kamer, W.Hans Jeroen, Louis-Martin Rousseau, Nadia Lahrichi, and Wim van Harten. A mathematical programming model for radiotherapy scheduling with time windows. In *Springer Proceedings of the Health Care Systems Engineering conference, 30 May-01 June, Montréal, Canada*, 2019.

Petra Vogl, Roland Braune, and Karl F Doerner. Scheduling recurring radiotherapy appointments in an ion beam facility. *Journal of Scheduling*, 22(2):137–154, 2019.

Min Wen, Emil Krapper, Jesper Larsen, and Thomas K Stidsen. A multilevel variable neighborhood search heuristic for a practical vehicle routing and driver scheduling problem. *Networks*, 58(4):311–322, 2011.

Johan Wittocx, Broes De Cat, and Marc Denecker. Towards computing revised models for fo theories. In *International Conference on Applications of Declarative Programming and Knowledge Management*, pages 85–101. Springer, 2009.

List of publications

Journal Articles

- Tu-San Pham, Pieter Leyman, and Patrick De Causmaecker. The intermittent travelling salesman problem. *International Transactions in Operational Research*, pages 525–548, 2018b.

IC (Papers at international scientific conferences and symposia, published in full in proceedings)

- Tu-San Pham, Louis-Martin Rousseau, and Patrick De Causmaecker. The Radiotherapy Scheduling Problem. In *Proceedings of MISTA'2019*, 2019b.
- Tu-San Pham, Jo Devriendt, and Patrick De Causmaecker. Declarative local search for predicate logic. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 340–346. Springer, 2019a.
- Tu-San Pham, Jo Devriendt, and Patrick De Causmaecker. Modelling Local Search in a Knowledge Base System. In *New Trends in Emerging Complex Real Life Problems*, pages 415–423. Springer, 2018a.
- Tu-San Pham and Patrick De Causmaecker. Variable Neighbourhood Search for the Intermittent Traveling Salesman Problem. In *Proceedings of META'2016*, volume 6, pages 57–65, 2016.

IMa (meeting abstracts, presented at international scientific conferences and symposia, published or not published in proceedings or journals)

- Tu-San Pham, Jo Devriendt and Patrick De Causmaecker, Formalize neighborhoods for local search using predicate logic, *DSO workshop colocated with CEC 2017 and CPAIOR 2017*. Padova, Italy, June 2017.

- Tu-San Pham and Patrick De Causmaecker, Knowledge representation and optimization, *21st conference of International Federation of Operation Research Societies*, IFOR 2017, Quebec, Canada, July 2017.
- Tu-San Pham and Patrick De Causmaecker, The intermittent traveling salesman problem, In *MISTA 2015 - Multidisciplinary International Scheduling Conference*, Prague, Czech Republic, August 2015.

AMa (Meeting abstracts, presented at other scientific conferences and symposia, published or not published in proceedings or journals)

- Tu-San Pham, Jo Devriendt and Patrick De Causmaecker, Formalize neighbourhoods for local search using predicate logic, *32th annual conference of the Belgian Operation Research Society*, ORBEL 32, Liege, Belgium, February, 2018.
- Tu-San Pham, Variable Neighbourhood Search for the Intermittent Traveling Salesman Problem, *PhD symposium Inria Lille Nord Europe*, Lille, France, June 2016.



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
CODES
Etienne Sabbelaan 53 box 7659
8500 Kortrijk
san.pham@kuleuven.be