Advanced modelling and search techniques for routing and scheduling problems

Charles Thomas

Thesis submitted in partial fulfilment of the requirements for the Degree of Doctor in Applied Sciences

January 2023

ICTEAM
Louvain School of Engineering
UCLouvain
Louvain-la-Neuve
Belgium

Thesis Committee:

Pr. Pierre **Schaus** (Advisor)	UCLouvain/ICTEAM, Belgium
Pr. Yves **Deville**	UCLouvain/ICTEAM, Belgium
Pr. Charles **Pecheur**	UCLouvain/ICTEAM, Belgium
Pr. Christian **Artigues**	LAAS-CNRS, France
Dr. Vinasétan Ratheil **Houndji**	Université d'Abomey-Calavi, Benin
Dr. Roger **Kameugne**	Université de Maroua, Cameroon

Advanced modelling and search techniques for routing and scheduling problems

by Charles Thomas

ICTEAM
UCLouvain
Place Sainte-Barbe, 2
1348 Louvain-la-Neuve
Belgium

This work was financed by the Walloon Region (Belgium) as part of the PRE-Supply project [Log].

Abstract

This thesis presents the application of several Constraint Programming (CP) techniques to combinatorial problems. In particular, hybrid scheduling and routing problems such as Dial-A-Ride Problem (DARP) are explored. A variant of this problem, the Patient Transportation Problem (PTP) is formalized and resolved. Various approaches to model the PTP and DARP are studied, including a scheduling model and a classical successor model. The usage of sequence variables to model the routes of vehicles is investigated. Two different implementations of a sequence variable are presented as well as several global constraints used in conjunction with these variables to provide efficient propagation algorithms. Additionally, the use of an adaptive variant of the Large Neighborhood Search (LNS) is considered in a black-box context, without prior knowledge about the problem being solved. The approach studied uses a portfolio of different heuristics combined with a selection mechanism to adapt the heuristics used to the current problem during the search. Experimental results show the efficiency of the techniques proposed and hint at promising research directions in the domain of PTP-like problems, sequence variables and Adaptive Large Neighborhood Search (ALNS).

Acknowledgements

Completing a Ph.D. is not an easy task. During my journey towards this goal, many people have been besides and behind me, encouraging me to persevere when facing doubts and supporting me through the difficult times. Thus, I would like to thank all the people that not only helped me making this thesis but also made it an incredible experience.

First I would like to thank my family and friends who supported me throughout this journey and gave me the strength to continue despite the difficulties. I would also like to thank all my friends and colleague from INGI who made the spirit and ambiance in the department such enjoyable and motivating. Among them, I would like to address my special thanks to Xavier, Hélène, Gorby, Mathieu, Fabien, Marie-Marie, Guillaume and Augustin with who I shared many great times. I would also like to extend my gratitude to the administrative and technical staff of the department who make everything run smoothly through hard work in the shadows; in particular, Vanessa, our secretary, without whom the department would drift into chaos in a matter of days if not hours.

Additionally, I extend my thanks to Roger, Quentin and Renata for the valuable scientific collaboration and enjoyable exchanges that we had. I would also thank the people from the PRESupply project and N-Side that made it an interesting challenge and provided me a great working environment. I would also like to thank my thesis committee and jury who guided me through this scientific journey and took the time to read my thesis.

Finally, I would like to give a huge thank you to Pierre, my advisor, who introduced me to the fields of combinatorial optimization and constraint programming; encouraged me to pursue a Ph.D.; backed me with his guidance, advices, ideas and support throughout this thesis; had kind and supportive words when needed and always believed in me even when that was not my case.

Contents

Introduction 1

1.1 Context

Nowadays, optimization is a large research field at the intersection of Mathematics and Computer Science. It deals with the resolution of large and difficult problems that arise in a variety of situations. Solving such problems by hand is resource consuming and sub-optimal if not sometimes impossible, thus driving the demand for efficient algorithmic approaches.

This thesis was financed by the Walloon Region as part of the PRESupply project [Log]. The aim of this research project was to provide small and medium businesses with affordable solutions to solve various optimization problems linked to supply lines and operations.

Among such problems, are routing and scheduling problems. Routing problems consist in constructing a path or tour in a given graph. This kind of problem often occurs in logistics or networking fields. A common example of such a problem is the Vehicle Routing Problem which consists in optimizing the transport of goods or people by a fleet of vehicles under various constraints. Some of these constraints can introduce scheduling aspects to the problem. Scheduling problems deal with the assignment of resources to tasks over time.

One problem that combines routing and scheduling characteristics is the Dial-A-Ride Problem (DARP). It consists in transporting users from one point to another. Pick-ups and drop-offs of users are constrained by time windows. This problem must be solved by numerous entities proposing on-demand transportation services. It is widely studied in the literature and various approaches have been proposed to tackle it. The Patient Transportation Problem (PTP) is a variant of the DARP that focuses on the transportation of people to medical appointments. This context adds several constraints to the original problem. This problem was provided by one of the partners of the PRESupply consortium. As part of this thesis, the PTP is studied and several techniques to solve it using the constraint programming paradigm are proposed.

Constraint Programming (CP) is an approach to solve combinatorial problems that consists in describing the problem to solve as a declarative model in terms of variables and constraints. Variables are elements of decision of the problem to which a value must be assigned. Constraints are relations be-

tween variables that limit the values that the variables can take in regards to each other. These relations can be used to propagate changes between variables when an assignation occurs using dedicated algorithms. A solution to the problem is found if all variables have a single value assigned such that the constraints are respected. A solver is used to perform a search on the space of possible solutions, using the constraints to reduce the search space to explore.

As the efficiency of the search depends on the quality of the model, it is important to be able to express a given problem with appropriate variables and constraints. Problems such as the PTP include decisions on how to sequence a series of elements or events. A dedicated sequence variable allows to directly represent such decisions in a model rather than using several integer variables linked together by constraints. Constraints designed to be used with sequence variables can leverage the structure and information of the variables to obtain gains in terms of propagation and performance. In this thesis, several implementations for sequence variables are proposed as well as a series of dedicated constraints useful to model problems such as the DARP and the PTP.

Besides the model, another important part of the CP paradigm is the search. Typically a backtracking search is used to explore the possible assignations of the variables in search of solutions. It consists in iteratively deciding values to assign to variables and using the constraint propagation to filter inconsistent assignations out of the remaining search space. If the current partial assignation leads to a violation of the constraints, a backtracking mechanism is used to revert to the previous decisions and explore alternative assignations. The order in which variables are assigned as well as their values can impact the propagation and thus the efficiency of the search. Heuristics are used to choose the order in which variables and values are assigned. In case of optimization problems, an objective function associated with a branch and bound algorithm guide the search towards better solutions. Advanced search strategies can be used to explore large search spaces efficiently.

One such search method is the Large Neighborhood Search (LNS). It uses a relax and rebuild approach that consists in iteratively relaxing parts of a solution to the problem and performing a search on the resulting search space in order to improve the current solution. The adaptive variant of this search method (ALNS) uses a portfolio of several relaxation and search heuristics called operators. At each iteration, a pair of operators is chosen and applied. The selection probabilities of the different heuristics are biased during the search based on their performances. The usage of this technique in a black-box context (without knowledge of the model) is studied in this thesis as well as the evaluation mechanism used to bias the selection of the heuristics.

This thesis is organized as follows: Chapter 2 presents the state of the art and introduces the notions needed to discuss the topics approached in the

rest of the thesis. Chapter 3 is focused on the ALNS search technique and its application in a black-box context. Chapter 4 discusses the Patient Transportation Problem and the methods used to model and solve the problem. In Chapter 5 the sequence variables is detailed as well as its implementations, constraints and application to the DARP and PTP problems. Finally, Chapter 6 concludes this thesis and discusses some of the future work possible on the topics presented.

1.2 Contributions

The contributions of this thesis are the following:

Adaptive Large Neighbourhood Search The Adaptive Large Neighborhood Search (ALNS) is an adaptive variant of the LNS search technique. It uses a portfolio of relaxation and search heuristics to adapt the search dynamically to the problem at hand. The application of this technique in a black-box context with a set of heuristics targeted at specific problems is examined (Chapter 3). A new evaluation mechanism that better deals with a set of heuristics presenting variable running times is also proposed (Section 3.2.3).

Patient Transportation Problem The Patient Transportation Problem consists in transporting patients to and back from medical appointments. This problem was proposed by the Centrale de Services à Domicile (CSD) [CSD], a non-profit organization based in Liège (Belgium) as part of the PRESupply Project. A formalization of the problem is proposed (Section 4.1) and two models are studied (Sections 4.4 and 4.3) as well as a dedicated search technique (Section 4.5).

Sequence Variables A sequence variable represents a set of elements to order. Two different implementations for a sequence variable are proposed and discussed. Both are based on an extension of the set variable that represents the set of elements to sequence along with an internal growing sequence that contains the elements already sequenced. The prefix sequence variable (Section 5.3) allows only to add elements at the end of the internal growing sequence. The insertion sequence variable (Section 5.4) allows new elements to be inserted at different points between sequenced elements and keeps track of possible insertion positions for unsequenced elements.

Furthermore, a series of constraints on sequence variables as well as their propagation algorithms for both implementations are proposed

(Section 5.5). The sequence variable and its constraints is used in models for the Patient Transportation Problem and the Dial a Ride Problem (Section 5.6).

1.2.1 Publications

These contributions were presented in several papers published at various conferences:

1. C. Thomas and P. Schaus. "Revisiting the Self-adaptive Large Neighborhood Search". In: *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. 2018, pp. 557–566.

   This paper presented at the CPAIOR 2018 conference presents the application of the ALNS technique in a black-box context, including the new evaluation mechanism.

2. Q. Cappart, C. Thomas, P. Schaus, and L.-M. Rousseau. "A Constraint Programming Approach for Solving Patient Transportation Problems". In: *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11008 LNCS. 2018. ISBN: 978-3-319-98333-2. DOI: 10.1007/978-3-319-98334-9_32.

   This paper presented at the CP 2018 conference presents the formalization of the PTP and the scheduling model proposed to solve the problem. A summary of the paper was also presented at JFPC19:

   C. Thomas, Q. Cappart, P. Schaus, and L.-M. Rousseau. "Une Approche de Programmation Par Contraintes Pour Résoudre Le Problème de Transport de Patients". In: *Actes Des 15es Journées Francophones de Programmation Par Contraintes JFPC 2019*. 2019, p. 31.

3. C. Thomas, R. Kameugne, and P. Schaus. "Insertion Sequence Variables for Hybrid Routing and Scheduling Problems". In: *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 12296 LNCS. 2020. ISBN: 978-3-030-58941-7. DOI: 10.1007/978-3-030-58942-4_30.

   This paper presented at the CP 2020 conference presents the insertion sequence variable, its implementation, two of its constraints and their usage on the DARP and PTP problems.

Additionally, the implementation of the algorithms described in this thesis is available in open-source as part of the OscaR CP library [Osc12].

Background 2

This chapter introduces the concepts and notions behind the topics discussed in the following chapters.

2.1 Combinatorial Problems

Combinatorial Problems can be described as finding an object among a finite, discrete collection of objects in order to satisfy given conditions. This object can take the form of an assignment, grouping or ordering of a subset of elements among a set of elements. Additionally, the goal may be to find an optimal object defined by an objective function. In this last case, it is called **combinatorial optimization**. Typically, the collection of objects is too large to enumerate and only described by either a concise representation or its properties. This makes it impossible to simply examine the objects one by one and selecting the best one. Hence, more efficient methods have to be used.

Combinatorial problems can be expressed as search problems, decision problems or optimization problems. A **search problem** is a problem for which any solution can be certified correct in a time bounded by a polynomial of the size of the input. In other words, while it may be complicated to find a solution to the problem, proving that such a solution is a correct one is easy. A **decision problem** can be formulated as answering the question "does a solution exist?" and an **optimization problem** to "what is the best solution?". While not technically the same these problems can usually be reduced to one another. **Reducing** a problem A to another problem B means that it is possible to solve A by transforming it to an instance of B, solving this instance and transforming the solution back to a solution of A.

Combinatorial problems can also be distinguished between static and dynamic problems. **Static** problems are fixed. Their input is known and stable. On the other hand, **dynamic** problems must deal with changes during their execution. The input of the problem, its objectives or conditions might change and have to be taken into account. For such problems, sometimes probabilities of changes are known in advance and can be used to mitigate them. Another possibility is to know in advance some possible change scenarios that can happen and can be used to plan contingencies. These problems are qualified

as **stochastic** problems.

Combinatorial Problems occur in many domains such as logistics, supply chain optimization, scheduling, planning, routing or propositional satisfiability. This class of problem is widely studied and many resolution methods have been proposed over the years. Often, real world problem exhibit characteristics and instance size which make it important to develop methods that are both efficient and easily adaptable.

2.1.1 Complexity

An important characteristic of combinatorial problems is whether or not the problem is "hard to solve" in a technical sense. In other terms, is it possible to devise efficient algorithms to solve the problem? The theory of **intractability** aims at answering this question.

The **computational complexity** (often referred simply as complexity) of an algorithm is a measure of the resources (time and memory space) needed to execute this algorithm. As the resources needed to run an algorithm or solve a problem are generally dependant on the input size N, the complexity is most often expressed as a function of the input size: $N \rightarrow f(N)$. Note that there might be several inputs of size n that exhibit different behaviors. Usually, we consider the worst possible case but alternatives are possible, such as using the average case. In most cases, finding a function $f(N)$ that exactly describes the resources used by an algorithm is not evident. Thus, the complexity is often given in terms of asymptotic bounds using a simpler function $g(N)$. Several notations are commonly used to express the complexity of an algorithm:

- The "big-Oh" notation $O(g(N))$ is used to express an asymptotic upper bound on the performance of an algorithm. The computational complexity of $f(n)$ is said $O(g(N))$ if there exists constants c and N_0 such that
$$|f(N)| \leq c|g(N)| \forall N > N_0 \tag{2.1}$$
- The "big-Omega" notation $\Omega(g(N))$ is used to express an asymptotic lower bound on the performance of an algorithm. The computational complexity of $f(n)$ is said $\Omega(g(N))$ if there exists constants c and N_0 such that
$$|f(N)| \geq c|g(N)| \forall N > N_0 \tag{2.2}$$
- The "big-Theta" notation $\Theta(g(N))$ is used in the case where both the upper bound and the lower bound are the same. The computational complexity of $f(n)$ is said $\Theta(g(N))$ if $f(N)$ is $O(g(N))$ and $\Omega(g(N))$.

Note that, as these notations are generally used to express bounds on the worst case performance of an algorithm, the performances could be better in practice than those indicated. Algorithms are often classified according to the type of function appearing in the O notation:

- **Constant** algorithms have a complexity of $O(c)$, where c is a constant. In other words, this is the class of algorithms which complexity does not depend on the input size N.
- **Logarithmic** algorithms have a complexity of $O(c \log_b(dN)$, where b, c and d are constants.
- **Linear** algorithms have a complexity of $O(cN)$, where c is a constant.
- **Linearithmic** algorithms have a complexity of $O(cN \log_b(dN)$, where b, c and d are constants.
- **Polynomial** algorithms have a complexity of $O(N^\alpha)$, where α is a constant > 1. This class can be further divided based on the polynomial into quadratic, cubic, ect. algorithms. Note that this last class is often used to refer to all the algorithms that have a running time less than exponential.
- **Exponential** algorithms have a complexity of $O(\alpha^N)$, where α is a constant > 1.

This last class is of particular interest as it means that the time required to solve the problem grows exponentially in relation with the input size. This implies that there will be an input size for which the algorithm will not return an answer in a reasonable amount of resources regardless of the speed of the computer that executes the algorithm.

Based on the computational complexity of algorithms, the complexity of a combinatorial problem is defined as the complexity of the best algorithm able to solve the problem. Note that we say that an algorithm solves a problem if it either finds a solution (or the best solution in case of optimization problems) or proves that such solution does not exists.

Combinatorial problems can be classified into complexity classes based either on the existence of an algorithm that solves the problem or being proven equivalent to another problem for which the complexity is known. The class of all the problems that can be solved in polynomial time is called P. These problems are considered as "easy" problems as in order to be qualified as polynomial, there must exists an algorithm that is proven to solve the problem in a feasible amount of time either directly or through reduction.

The name NP is commonly used to describe the class of all the problems for which a solution can be verified in polynomial time. These are problems

for which finding a solution is not guaranteed polynomial but proving that such solution is correct is polynomial in regards to the input of the problem. NP stands for **Non-deterministic Polynomial**. As explained in [Sip12], the **non-determinism** is the theoretical notion that an algorithm may provide several possible actions for any given situation. When encountering multiple possibilities of execution, a non-deterministic algorithm would lead to several parallel states. If any of these parallel executions path leads to a solution, we consider that the non-deterministic algorithm solves the problem. Conceptually, it is as if at each choice, the algorithm is able to "guess" the option that leads to a solution. Another way to conceptualize non-determinism is to consider that whereas a deterministic algorithm follows a single "computation path", a non-deterministic algorithm would yield a "computation tree" which has at least one branch that leads to a solution. If a problem can be verified in polynomial time, with the help of non-determinism we could solve this problem in polynomial time by "guessing" the solution and then verifying it. Thus, the class NP can be defined more precisely as the class of all the problems which can be solved in polynomial time by an algorithm using non-determinism.

Note that while $P \subseteq NP$, whether these two classes of problems are equivalent or different has never been proven despite the strong intuition that $NP \neq P$. In other words, so far, no single problem has been proven to be part of NP but not P. This is a fundamental question in the field of computer science as it means that such problems would be proven not solvable in polynomial time and thus, the search for polynomial algorithms for these problems could be abandoned. On the other hand, if $P = NP$ is proven, that would mean that potentially, every problem in NP can be solved in polynomial time.

Definition 1. A problem A is said to **reduce to** another problem B if we can use an algorithm that solves B to develop an algorithm that solves A.

Note that reducing a problem to another may bring some extra cost which can be quantified by its complexity.

Definition 2. A problem is said to be ***NP*-complete** if

1. this problem is in NP;
2. all problems in NP can be polynomially reduced to this problem.

This means that if we can efficiently solve this problem, we could efficiently solve any problem in NP. Since most researchers in the field believe that $P \neq NP$, it is expected that a polynomial algorithm cannot be found for any NP-complete problem.

Finally, we say that a problem is ***NP*-hard** if the second condition of *NP*-complete is satisfied. Thus, for *NP*-hard problems, we do not know if the problem is part of *NP* but we know that all problems in *NP* can be polynomially reduced to this problem. A lot of combinatorial problems that appear in practice have either been classified as in *P* or as *NP*-hard while for some the question remains open. This knowledge is important as it directs the resolution methods that can be used to tackle a given problem. More details on the complexity of combinatorial problems can be found in [Kar72; Kar75; Tar78; AB09; SW11; DK11]

2.1.2 Resolution Methods

There exists diverse techniques to tackle combinatorial problems. These techniques can roughly be divided into two categories: **exact** methods consist in enumerating solutions by exploring the whole search space (parts the are proven to not contain a (better) solution can be discarded). The advantage of this approach is that there is a certainty that the problem will eventually[1] be solved. In case of optimization problems, exact methods can prove that the solution found is optimal. Exact methods include:

- **Constraint Programming (CP)** which is the approach that is used in this thesis. This approach consists in expressing the problem to solve as a declarative model. The specificity of this paradigm compared to other declarative methods is that the components used in the model do not specify algorithmic steps but rather describe properties that a solution must have. Another advantage of CP over other approaches is that it uses generic modeling components that can easily be combined to tackle different problems. The CP Paradigm is described in details in Section 2.2

- **Linear Programming (LP)** [DT03; MG07] consists in representing the problem in terms of linear relationships. Then, an optimization technique is used to find optimum in the search space. Techniques to solve linear programming problems include **Lagrangian Relaxation (LR)**, **Integer Programming (IP)** [Wol20] and **column generation** [DDS06; DPR06].

- **Dynamic Programming (DP)** [AM07] is a technique based on the decomposition of problems in smaller, simpler problems. These subproblems are then solved and their solutions used to solve larger problems.

[1] "Eventually" might be a (very) long time!

- **Satisfiability (SAT)** [BHv09] refers to the field of satisfiability problems. Combinatorial problems can be reformulated as SAT problems and solved with dedicated techniques.

- **Multi-valued Decision Diagrams (MDD)** are a specific form of Decision Diagram (DD). They consist in a layered graph that encodes sets of decision sequences as paths that join a source node to a terminal node. They can be used to combine dynamic programming with a compact encoding [BHH11; Ber+14; GSC21], allowing to solve large problems that would take too much space in a classical dynamic program.

Sometimes problems to solve have a search space too large to explore in a reasonable amount of time. This is where the second category of methods are privileged. These techniques are called **approximate** algorithms or sometimes **(meta)heuristic** techniques [GP10]. They consist in foregoing the complete exploration of the search space in order to find faster solutions that are close to optimal. Approximate techniques include **Local Search (LS)** [AAL03] techniques that iteratively improve a solution by incremental modifications and **machine learning** [Rob14; Bon17]. This approach consists in training a model based on historic data to solve the problem. It is used in many fields of computer science and has been successfully applied on combinatorial problems [BLP21; Maz+21].

Note that many of these methods can be combined or even hybridized. There exists many more approaches to solve combinatorial problems. For more information on this subject, see [KS99; Vaz03; MR22].

2.1.3 Traveling Salesman Problem

A typical combinatorial optimization problem is the Traveling Salesman Problem (TSP) [HPR13]. Given a set of N cities and the distance between each pair of cities, the problem consist in finding the shortest path that visit each city exactly once and ends in the depart city. More formally, the problem can be generalized as finding the shortest **Hamiltonian cycle** on a graph G defined as $G = (V, E)$ where V is a set of vertices and E a set of edges, each associated with a cost expressed either as a function or in a distance matrix. Figure 2.1 shows an example of TSP problem along with a possible solution.

In many real-world applications of this problem, the graph represents a transportation network. Each vertex $v \in V$ corresponds to a location to visit and each edge $e \in E$ represents the trip between two locations and is associated with a cost. Some arcs may be directed in case of differences of costs between both directions. Note that even if the initial network provided may be incomplete, it is possible to build a complete graph by taking into account the shortest path between the concerned nodes on the original network.

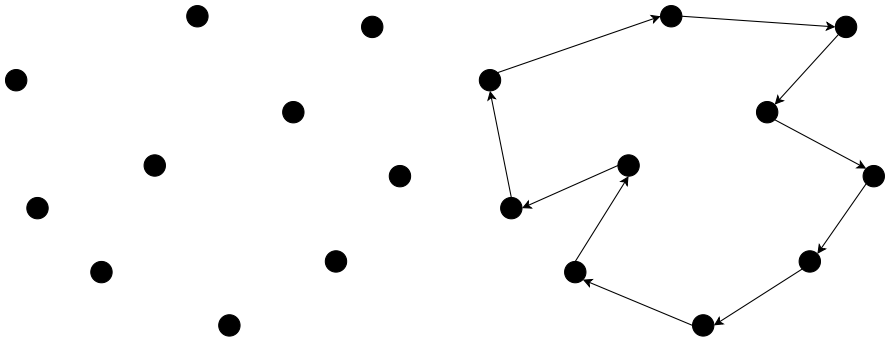

Figure 2.1: An example of TSP instance (left) and a possible solution (right)

The TSP has been proven NP-complete by Karp in 1972 [Kar72]. This property is especially important as many problems in routing can be reduced to a TSP. Another interesting property is the **triangle inequality** [JK89] which states whether or not the cost of an arc $cost_{a,b}$ between two nodes a and b must be shorter or equal to the cost of any path going through a third node c:

$$cost_{a,b} \leq cost_{a,c} + cost_{c,b}; \forall a, b, c \in V \tag{2.3}$$

This property may impact the algorithm used to solve such problems and thus is important to take into account.

The traveling salesman problem arises in many cases [LK75] and is also the basis for many more advanced graph-based problems. It is thus widely studied in the literature. Many variants have been defined over the years [Dum+95; AFG01; Lóp+13; IJ14; SS15; Tod+17; Boc+21] and various approaches have been proposed to tackle this problem [LSD90; CP80; ZL10; ONZ21; Cap+21; Red+22]. More details on this problem and its applications can be found in [App+11; Dav10; Gre08; Näh11].

2.1.4 Vehicle Routing Problem

The Vehicle Routing Problem (VRP) [TV02a] expands the TSP by considering a fleet of **vehicles** that must visit a set of **locations** in order to deliver goods or provide services. Similarly to the TSP, the problem considers a transportation network expressed as a graph $G = (V, E)$. One of the vertices represents the **depot** where each vehicle must start its route and end it. Figure 2.2 shows an example of VRP problem with three vehicles. The depot is the central vertex.

Many variants of the problem exist [EVR09; ITV14; BRV16], each with their own constraints and objectives. Common variants of the problem include:

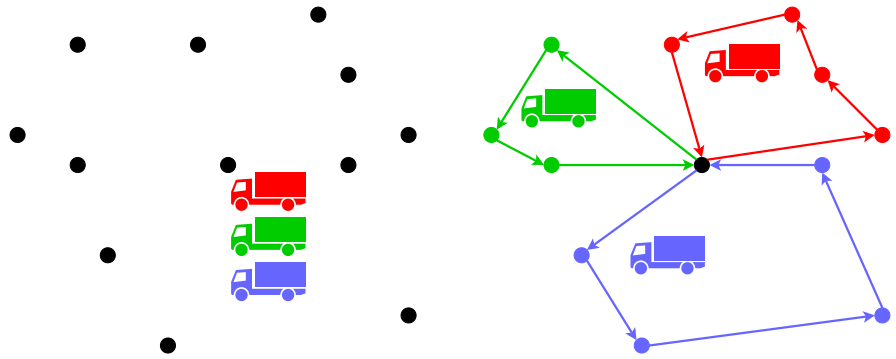

Figure 2.2: An example of VRP instance with three vehicles (left) and a possible solution (right)

- **The Capacitated VRP (CVRP)** [TV02b; Ral+03] which adds capacity constraints to the vehicles;
- **The Vehicle Routing Problem with Simultaneous Pick-up and Delivery (VRPSPD)** [KLT20] where goods need to be picked or dropped at various locations;
- **The Vehicle Routing Problem with Time Windows (VRPTW)** [BG05b; BG05a] which adds scheduling constraints that associate the visits with time windows in which they must be made;
- **The Multi-Depot Vehicle Routing Problem (MDVRP)** [Mon+15] where more than one depot exists;
- **The Dynamic Vehicle Routing Problem (DVRP)** [Pil+13] where dynamic changes to the problem have to be handled during the search. This variant may incorporate stochastic information on possible future changes [RPH16].

Note that these variants are not exclusive and can be combined together.

Typically, the objective is to minimize the cost of operations which can be based on the length of the routes or the number of vehicles required. Other objectives may consist in minimizing delivery times or maximizing some profit or score based on the deliveries performed. There also exist variants with multiple objectives combined [JST08]. More details on this problem and its resolution methods can be found in [Lap92; Cor+05; GRW08; TV14].

2.1.5 Dial-A-Ride Problem

The Dial a Ride Problem (DARP) is a variant of the VRP that consists in routing a fleet of vehicles in order to transport clients from one place to another. As

the problem deals with transportation of people, a minimum service quality must be assured. Indeed, unlike goods, passenger do mind if they have to wait too long for their ride or in transit. Hence, additional constraints such as maximum ride duration and service time windows must be enforced.

It is a problem widely studied in the literature and several variants exist: the fleet can be composed of several vehicles [CL03a] that can be heterogeneous [Par11], users can have different characteristics [Par+12], availability of vehicles can be constrained [Psa83], patients can require a return trip [MIM07], several depots can be present [CGL97], alternative routes between locations of the road network may be considered [Gar+10], etc. A large scope of objective functions can also be considered such as minimizing the waiting time of users or maximizing the number of accepted requests. Multi-objective approaches have also been introduced [Par+09]. Besides, the problem can either be solved offline [Ber+07] or online [Att+04]. In the former case, all the requests are known in advance whereas they appear gradually in real-time in the latter. Aforementioned references are only few examples of the broad literature dedicated to DARPs. A good summary of the different variants and methods was nevertheless proposed by Cordeau and Laporte [CL07]. More recent developments can be found in [MBC17].

The variant discussed in this thesis was proposed by Cordeau and Laporte [CL03b]. Note that the set of all passenger requests is assumed to be known in advance and fixed.

Formally, the DARP is defined on a complete graph $G = (X, E)$ where $X = \{X_0, \dots X_{2n}\}$ is a set of vertices, each corresponding to a specific location or stop and E a set of edges, each corresponding to a possible travel from one location to another. The first vertex $X_0 \in X$ corresponds to the depot. To each stop $i \in X$, is associated a time window $[ea_i, la_i]$, a service duration srv_i and a load ld_i. These parameters are fixed for the depot stop X_0 or $i = 0$ at $[ea_0 = 0, la_0 = T_{max}]$, $srv_0 = 0$, $ld_0 = 0$ where T_{max} is the planning time horizon. A transition matrix $trans_{i,j}$ indicates for each edge $(i, j) \in E$ the non-negative travel time $(trans_{i,j})$ from location i to j, assumed to satisfy triangle inequality:

$$trans_{i,j} \leq trans_{i,k} + trans_{k,j}; \forall i, j, k \in X \tag{2.4}$$

Let R be the set of requests of size n. Each request $r \in R \mid 1 \leq r \leq n$ is associated to a pair of stops X_r and X_{r+n} that correspond to the *pickup* and *drop* of the request. The load for the pickup stop is always strictly positive $ld_r > 0$. For the drop stop, it is the opposite of the pickup stop load $ld_{r+n} = -ld_r$ and thus negative. A stop is critical if its time window is restricted and non critical if it is initialized to $[0, T_{max}]$. A request is either *inbound* or *outbound*. For inbound requests, the pickup stop is critical whereas the drop stop is non critical. For outbound requests the pickup is non critical and the drop is critical.

A global parameter R_{max} indicates the maximum ride time (the difference between the arrival at the drop and the departure at the pickup) of any request.

Let V be the set of vehicles. There are m vehicles available. Each vehicle $v \in V$ has the same profile, defined by a capacity K and a maximum route duration D_{max}.

The objective is to minimize the total routing cost of the vehicles (defined as the total distance traveled by the vehicles) (0) under the following constraints:

(1) Each route begins at the depot X_0;

(2) Each route ends at the depot X_0;

(3) The load of any vehicle never exceeds its capacity K;

(4) For each pair of stops serviced by a same vehicle (i, j), the difference between the arrival time at the second stop (j) and the departure at the first stop (i) is higher or equal to the travel distance between the two stops $trans_{i,j}$;

(5) For each stop i, the service of the stop starts inside its time window $[ea_i, la_i]$;

(6) A pickup stop X_r is always visited before the associated drop stop X_{r+n};

(7) A pickup stop X_r and its associated drop stop X_{r+n} are both serviced by the same vehicle;

(8) For each stop i, the service time to embark or disembark the vehicle srv_i is respected;

(9) The ride time between a pickup stop X_r and its associated drop stop X_{r+n} never exceeds the maximum ride time R_{max};

(10) The total duration of any route never exceeds the maximum route duration D_{max};

(11) Each request is serviced.

2.2 Constraint Programming

Constraint Programming (CP) [RVW06] is a programming paradigm used to solve combinatorial problems. In CP, the problem is expressed as a declarative model in terms of variables and constraints. Then, a backtracking search is used to explore the possible solutions of the problem. The paradigm stems from research in the 1960's by Sutherland [Sut63] and was developed during

the following decades until being formally defined in 1980 by Steele [Ste80]. Since then, CP has been successfully used in many domains such as vehicle routing [Sha98; BB19], planning and scheduling [VC99; Tim02; Sim+15], networks [BPA01; Bar+07; Har+15a], bioinformatics [GBY01; BK08; All+14], datamining [DGN10; Bes+16] and recently machine learning [VH15; Ver+20]. Besides its proven efficiency over other methods [Lab18a; Mal+18] on a range of frequent problems, another advantage of CP is versatility and adaptability. Due to its nature, a CP model can easily be modified to tackle variations of a problem. Finally, CP can also be used in conjunction with other approaches such as local search [Bac+00] MIP [HB12; SW12; Tan+19] or SAT [OSC09; Stu10; Art+14].

2.2.1 Model

The model is a representation of the problem using **variables** to which a value must be found tied together by **constraints**.

A variable is an element of decision of the problem. Each variable is associated to a **domain** which is a set of values that the variable can take. Tying a variable to a single value of its domain is called an **assignment**. Solving a constraint problem consists in finding a valid assignment for each variable. A **solution** is thus a state where each variable is assigned to a single value such that the constraints are respected. Note that several solutions are often possible.

A constraint is a mathematical condition that must be satisfied in order for a solution of the problem to be valid. Expressed in the CP paradigm, a constraint is relation between several variables that limits the values assigned simultaneously to the variables.

Expressing a Combinatorial Problem with the CP paradigm In constraint programming, a combinatorial problem is expressed as a Constraint Satisfaction Problem (CSP) or a Constraint Optimization Problem (COP).

The goal of a CSP is to find a solution that satisfies all the constraints. Alternatively, one might want to retrieve all or several possible solutions to the problem or simply find if the problem is feasible. Note that proving that a CSP is feasible is done as soon as a solution is found but proving that a problem is infeasible requires to explore completely the search space without finding any solution.

Definition 3. A Constraint Satisfaction Problem (CSP) is a triplet (V, D, C) consisting of: a set of variables V, each associated to a domain in D and a set of constraints C restricting the domains of the variables.

Definition 4. A solution is a state of the CSP such that each variable $x \in V$ is assigned to a single value $v \in D_x : D_x = v$) and each constraint $c \in C$ holds.

Example 2.2.1. Let us consider a very simple CSP: $x, y, z \in \{1, 2, 3\}, x+y = z$

- The variables are x, y, z;
- the domains are: $D_x = D_y = D_z = \{1, 2, 3\}$;
- there is a single constraint $x + y = z$.

The possible solutions for this CSP are: $\{(x = 1, y = 1, z = 2), (x = 2, y = 1, z = 3), (x = 1, y = 2, z = 3)\}$.

Example 2.2.2. Another example of CSP is the N-Queens problem. It consists in placing N queens on a chessboard of size $N \times N$ so that they are unable to attack each other. The problem can be formulated as such: For each queen $i \in N$, we have a variable Q_i that indicates the column in which the queen at row i will be placed (only one queen can be on any row). We have the following constraints:

- Any pair of two queens cannot be in the same column:

$$\forall i, j \in N \mid i \neq j : \ Q_i \neq Q_j \tag{2.5}$$

- Any pair of two queens cannot be in the same diagonal:

$$\forall i, j \in N \mid i \neq j : \ |i - j| \neq |Q_i - Q_j| \tag{2.6}$$

  where $|x|$ denotes the absolute value of x.

A possible solution for this problem with $N = 8$ is illustrated in Figure 2.3. Notice that a same problem can be formulated as different CSPs. For this problem, we could have a boolean variable for each square of the chessboard that indicates the presence of a queen in this square. We could also replace the set of binary constraints on each pair of queens by two *Alldifferent* constraints (this constraints ensures that a set of variables have each a distinct value, see Section 2.2.1.2).

Example 2.2.3. Finally, a well known example of CSP is the sudoku problem illustrated in Figure 2.4. It consists in placing numbers in a 9 by 9 grid such that all the numbers in a same row, column or 3 by 3 sub-grid are different. The problem can be formulated as such: For each square $c_{i,j}$ of the grid, we have a variable $v_{i,j}$ of domain $D_{i,j} = [1; 9]$ that indicates the number contained in the square. The constraints are:

- For each row $i \in [1; 9]$: *Alldifferent*$(c_{i,j}, \forall j \in [1; 9])$
- For each column $j \in [1; 9]$: *Alldifferent*$(c_{i,j}, \forall i \in [1; 9])$

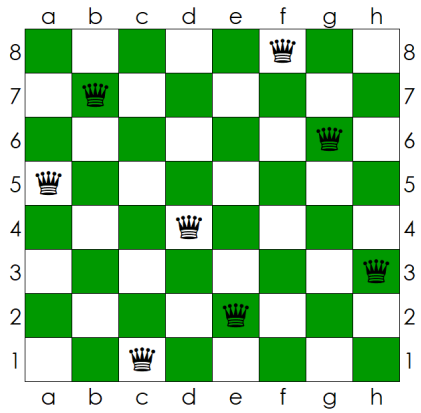

Figure 2.3: Example of 8-Queens problem (source: [Enc19])

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figure 2.4: Example of sudoku (source: [Tim17])

- For each sub-grid $s_{k,l} \mid k, l \in [1;3]$: $Alldifferent(c_{k\cdot m, l\cdot n}, \forall m, n \in [1;3])$

A constraint optimization problem (COP) is an extension of a constraint satisfaction problem to which an objective is added that indicates the quality of a solution to the problem. The objective function associates one or several variables of the problem to a value that must be maximized or minimized.

Definition 5. A Constraint Optimization Problem is a quadruplet (V, D, C, O) where (V, D, C) defines a CSP and O is an objective function.

Example 2.2.4. If we add the objective function $O = \max.\ x+z$ to the CSP of Example 2.2.1, we obtain a COP. The optimal solution is $(x = 2, y = 1, z = 3)$ with an objective value of 5.

An optimal solution maximizes or minimizes the objective function. Solving a COP to optimality consists in proving that the best solution obtained is optimal. Generally, that implies completely exploring the search space which may not be feasible in a lot of cases.

2.2.1.1 CP Variables

A CP variable is characterized by its domain which is used to model some decision in the problem. Conceptually, the domain is a set of possible states that the variable can take. An important characteristic of CP variables is that in addition to represent their domain, they must also do so in a reversible way. Indeed, during the CP search, the domain of the variables will be reduced (possible values will be removed). However, when encountering a violation of a constraint or to explore other parts of the search space, these changes

must be reverted and the domains restored to a previous state. Thus, the variables must implement mechanisms to revert their domain. Several possible strategies are possible and are discussed in Section 2.2.2.3.

There exists several types of variables in CP, each having specific domains. While Integer variables are the most frequently used, other variable representation can allow a better modeling of the problem and may even be necessary to model more complex problems.

Boolean Variable Boolean variables represent binary decisions They can only take on of two values: *true* or *false*. They are often used in conjunction with logical constraints.

Integer Variable An integer variable represents an unknown number to which a value must be chosen among a set of possible values. Its domain is thus a set of integers.

Interval Variable An interval variable [IBM21] (also sometimes referred as an activity variable) represents an interval of time during which an event occurs (a task is performed, a resource is used,...). It is characterized by a start s, an end e and a duration d, each of them having a range of possible integer values. The duration corresponds to the difference between the start and the end values: $d = e - s$. Additionally, a boolean x might indicate whether the activity takes place or not. In this case it is called a *conditional interval variable*. The variable is fixed once each of its attributes is assigned to a single value. Such variables are often used in scheduling problems.

Example 2.2.5. For example, let us consider an interval variable representing an activity with the following domain: $start = [0; 2]$, $end = [6; 9]$, $duration = [4; 9]$. As illustrated in Figure 2.5, the activity could begin at any time between 0 and 2, its length would be of minimum 4 and maximum 9 and it would end between 6 and 9.

Note that this kind of variable can be modeled with three separate integer variables (corresponding to the start, end and duration) and a boolean variable linked together by constraints. Using an interval variable over integer variables provides a better modeling and might allow to use the problem structure for gains in terms of propagation. However, it can also lead to unnecessary complexity and requires support in the solver as well as dedicated constraints.

Set Variable A set variable represents a set of elements. It is used to model situations where one must decide which elements will be part of a set. It domain consists in a set of elements (often represented by integer values) that

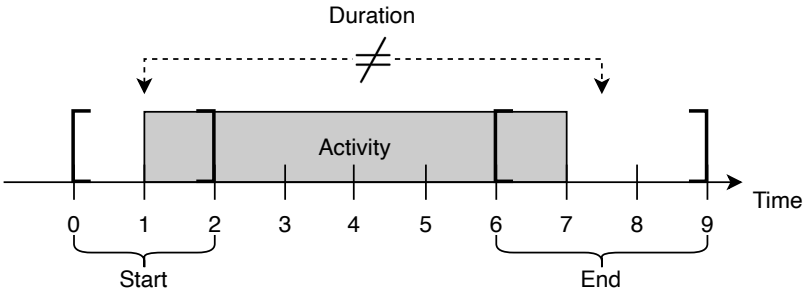

Figure 2.5: Illustration of an interval variable

have two possible states: part of the set (denoted *required*) or excluded from the set (denoted *excluded*). Initially each element state is undecided (denoted *possible*). When each element has its state fixed, the variable is bound. Finding an assignment for a set variable thus corresponds to choosing a subset of the original set of elements in the domain.

In formal terms, the variable can be defined as so: Let $X = \{0, ..., n\}$ be a finite set and $\mathcal{P}(X)$ the set of subsets (power set) of X. The inclusion $\subseteq$ relation defines a partial order over $\mathcal{P}(X)$ and the structure $(\mathcal{P}(X), \subseteq)$ is a lattice generally used to represent the domain of a finite set variable. To avoid explicit exhaustive enumeration of set domain, three disjoint subsets of X are used to represent the current state of the set domain (see [Ger97]). The domain is defined as

$$\langle P, R, E \rangle \equiv \{S' \mid S' \subseteq X \wedge R \subseteq S' \subseteq R \cup P\} \tag{2.7}$$

where P, R, and E denote respectively the set of Possible, Required and Excluded elements of X. At any time we have that P, R and E form a partition of X. The domain represents a powerset lattice. The variable S with domain $\langle P, R, E \rangle$ is bound if P is empty. Table 2.1 contains the supported operations on a set variable S of domain $\langle P, R, E \rangle$ with their complexity. Other work on this kind of variable includes [Ger94; GV06; YH11; YH09; YV10].

Table 2.1: Operations supported by set variables

Operation	Description	Complexity
`requires`(S, e)	move e to R, fails if $e \in E$	$\Theta(1)$
`excludes`(S, e)	move e to E, fails if $e \in R$	$\Theta(1)$
`isBound`(S)	return true iff S is bound	$\Theta(1)$
`is{Possible/Required/Excluded}`(S, e)	return true iff $e \in \{P/R/E\}$	$\Theta(1)$
`all{Possible/Required/Excluded}`(S)	enumerate $\{P/R/E\}$	$\Theta(\lvert\{P/R/E\}\rvert)$

Sequence variables A sequence variable represents a sequence of elements or events. It can be seen as associating a set of elements with an ordering between these elements. The domain of a sequence variable *Seq* over a set of elements X is the set of all possible sequences that can be made from the elements of X. Sequence variables are discussed in details in Chapter 5.

Other type of variables In addition to the variables presented above, there exists other types of variables. For example, graph variables [HM98; DDD05] are used to represent graphs and model graph-based problems. Continuous variables [Wal02; JB04] represent a continuous interval of values and are used to model continuous problems [YH06]. However, as these variables are not discussed in this thesis, they are not detailed.

2.2.1.2 CP Constraints

With variables, constraints are the building blocks of CP models. Their role is ensuring that the domains of the variables are consistent in regards to the model. This is done through the propagation.

Propagation Constraint propagation occurs at the beginning of the search and is triggered again when the domain of a variable is modified. The constraints that include the affected variables are notified of the modifications and can use this information to propagate changes to the domains of other variables that are related through the constraint. This can in turn lead to more domain modifications which are propagated and so on. This process continues until no more change is possible or an inconsistency is detected. In the first case, we say that a **fixed point** has been reached. The propagation phase stops and the search continues. In the latter case, a backtrack is triggered: the latest decision is reverted and the domains are restored to their previous state.

Example 2.2.6. Let us consider three variables: $x = \{1, 2, 3\}$, $x = \{2, 3\}$ and $z = \{3, 4, 5\}$ under the constraints $x < y < z$. If the value 1 is removed from the domain of x , thus reducing its domain to $\{2, 3\}$, the propagation is triggered. Since the minimal value for x is 2, any value ≤ 2 can be removed from the domain of y resulting in $y = \{3\}$. As the domain of y is reduced the propagation is triggered again resulting in the removal of 3 from the domains of x and z. The propagation is triggered again for each domain reduction but no change can be deduced thus ending the process. The resulting domains are: $x = \{2\}$, $y = \{3\}$ and $z = \{4, 5\}$.

An algorithm responsible for the propagation is called a **filtering algorithm**. It is responsible for reducing the domains of the variables and detecting inconsistent partial assignments by ensuring that the domains are con-

sistent. Generally constraints implement one or several filtering algorithms that are called depending on the type of changes that occurred.

Local Consistency Ideally, the propagation would achieve a fully consistent state for all the domains with all the constraints (called **global consistency**). However, in practice, such result is often unreachable. Filtering algorithms often have a limited knowledge of the whole model as their scope doesn't include variables that are not directly affected or other constraints. Usually, reaching global consistency is itself expensive in terms of complexity or even a *NP*-complete problem, making some filtering algorithms too costly. Thus, through propagation, we aim at reaching some form of **local consistency**. The notion of local consistency can be informally defined as having some parts of the domains in a "desired form", meaning that though global consistency is not necessarily achieved, some parts of the CSP are consistent at some level. Note that local consistency does not necessarily implies global consistency.

Therefore, different local consistencies have been defined. Here are some of the most commonly used notions:

- **Node Consistency** is the most basic form of consistency.

  Definition 6. A CSP is said to be node consistent if for every variable $x \in V$, every unary constraint (see next paragraph) on V is satisfied.

- **Arc Consistency** is reached for a binary constraint (see next paragraph) if every value in both domains is part of a solution.

  Definition 7. A binary constraint c on two variables $x, y \in V$ of domains D_x and D_y is called arc consistent if

  $$\forall k \in D_x : \exists l \in D_y \mid (k,l) \in c \wedge \forall l \in D_y : \exists k \in D_x \mid (k,l) \in c \quad (2.8)$$

  where the notation $(k,l) \in c$ indicates that the constraint c holds for the values k and l. A CSP is arc consistent if all of its binary constraint are arc consistent.

  We say that a value $v \in D_x$ has a **support** $s \in D_y$ for a constraint c if the pair (v,s) is valid under c.

- **Generalized Arc Consistency (GAC)** generalizes the notion of arc consistency to global constraints (with more than two variables, see Section 2.2.1.2). It is also called **hyper-arc consistency**.

Definition 8. A constraint c on a series of variables $x_1, \ldots, x_k$ is said to be hyper-arc consistent if

$$\forall x_i \mid i \in [1; k], \forall v \in D_i : \exists d \in c \mid v = d_{x_i} \tag{2.9}$$

where the notation $d \in c$ indicates that the constraint c holds for the assignment d and the notation $v = d_{x_i}$ indicates that the variable x_i takes the value v in the assignment d. A CSP is hyper-arc consistent or GAC if all its constraints are hyper-arc consistent.

In other words, a constraint is hyper-arc consistent if each value in the domain of each variable is part of an assignment that is valid under the constraint.

There exists other definitions of consistency such as path consistency, k-consistency or bound consistency. For more details, see [Apt03].

Stronger local consistency can lead to a larger reduction of the search space but this often comes at a cost in term of resources used by the filtering algorithms. On the other hand, despite leading to a larger search space and being less effective at detecting inconsistent assignments, weaker local consistency can often be reached with fast algorithms. Thus, there is a trade-off between the resources required by the filtering algorithms and the local consistency achieved. Often, these two aspects must be balanced in order to achieve a speed up of the resolution process.

Types of Constraints Constraints can be categorized in several categories:

- **Unary Constraints** are constraints that affect a single variable. For example, the constraint $x \geq v$ where x is a variable and v a value.
- **Binary Constraints** affect only two variables. For example, the equality constraint $x = y$ where x and y are variables.
- **Extensional Constraints**, also called table constraints, are defined by enumerating the set of all combinations of values that are allowed (called supports) or forbidden (called conflicts), see [Ver21].
- **Arithmetic Constraints** are defined by expressions such as $=$, $\neq$, $<$, $>$, $\leq$, $\geq$, ...
- **Logical Constraints** are defined by logical expressions (not, and, or, implication, ...).

- **Global Constraints** are defined over a non-fixed number of variables with an explicit semantic. These are often redundant as they encapsulate a set of simpler constraints. However they are useful as they ease the modeling of a problem and can leverage this global view of the problem for a more efficient propagation. Global constraints can be defined in more formal terms as a constraint that models a set of constraints of the same type. For example, the constraint $\mathit{Alldifferent}(x_1, x_2, x_3)$ represents the set of binary constraints $\{x_1 \neq x_2, x_1 \neq x_3, x_2 \neq x_3\}$. Note that in the case where $D(x_1) = D(x_2) = D(x_3) = \{1, 2\}$ while the binary inequality constraints individually hold, the global $\mathit{Alldifferent}$ constraint can detect an inconsistency.

Notice that these categories are not necessarily exclusive. For example, the constraint $x \leq y$ (where x and y are variables) is both binary and arithmetic. The following paragraphs details several constraints that are used or referred to further in this thesis.

Alldifferent The *Alldifferent* constraint [Lau78; Bel14a] is a global constraint which states that a set of variables must take distinct values. The constraint can be decomposed into set of binary *difference* constraints. However, a global propagation is more efficient and can achieve stronger consistencies. Several propagation algorithms are detailed in [R197; Lec96; Pug98; MT00; Lóp+03]. Note that the complexity of ensuring that this constraint is GAC is polynomial.

Sum The *Sum* constraint [Bel14f] links a variable s to a set of variables X such that the value of s is the sum of the values of the variables in X: $s = \sum X$.

Element Given two variables x and y and an array of integers Z, the *Element* constraint [VC88; Bel14d] enforces the relation $Z_y = x$. In other words, the constraint makes sure that the variable x takes the value at index y in Z.

Circuit The *Circuit* constraint [Lau78; Bel14b], also sometimes referred as *Predecessor/Successor* constraint is used to model situations where a set of variables represent a graph G, each variable corresponding to a node in G and its domain to a set of successor nodes to which the variable points. The constraint enforces a Hamiltonian circuit on this graph. Its domain is a set of integer variable with values corresponding to the index of the next variable in a tour visiting all of them. Figure 2.6 shows the assignation corresponding to the tour $[3, 2, 5, 4, 6, 1, 0]$.

The constraint is often used in routing problems to model routes of vehicles. It is related to the *Alldifferent* constraint as it must hold in order to

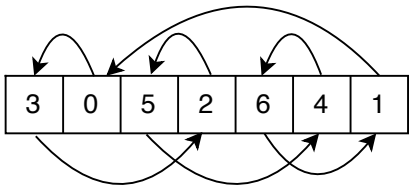

Figure 2.6: Valid assignation of an array of variables under the *circuit* constraint

satisfy *circuit*. Thus, the *circuit* constraint often uses *Alldifferent* propagation algorithms in conjunction with sub-tour elimination. Several of its propagation algorithms are detailed in [Hoo12; FS14]. Notice that in contrast to other global constraints such as the *Alldifferent* constraint, ensuring that the *Circuit* constraint is GAC is NP-hard as it is the same as solving the Hamiltonian cycle problem [GJ79].

Precedence The *Precedence* constraint [Bel14e] is used in scheduling problems to enforce the precedence of one or several event(s) over another. Given a set of n tasks T, each pair of consecutive tasks should be ordered: $T_i \prec T_{i+1}, \forall i \in [1; n-1]$ As such, it is mainly used in conjunction with interval variables.

Cumulative The *Cumulative* constraint [AB93; Bel14c] is also mainly used in scheduling and routing. Given a set of tasks $A_{1...n}$(that can be represented by interval variables) and associated loads ld representing consumption of a resource during the lapse of the interval, it ensures that the cumulative use of the resource never exceeds a given total capacity. This behavior is illustrated in the arbitrary example of Figure 2.7. We consider 4 tasks (A_1, A_2, A_3 and A_4) and a capacity K of 3. The task A_3 has a load of 2 while the other have a load of 1. The bottom part shows the execution of the tasks. The top part corresponds to the **load profile**, which is a representation of the total load used at any time. As we can see, in this case, the load profile never exceeds the capacity K, thus the constraint holds.

Many filtering algorithms have been proposed for this constraint over the years [Abd82; CL96; Vil07; MV08; SW10; Bel+11; Kam+14]. Note that this constraint can also be referred as *NoOverlap* when the capacity and loads are set to 1, ensuring that any task cannot take place simultaneously with any other task.

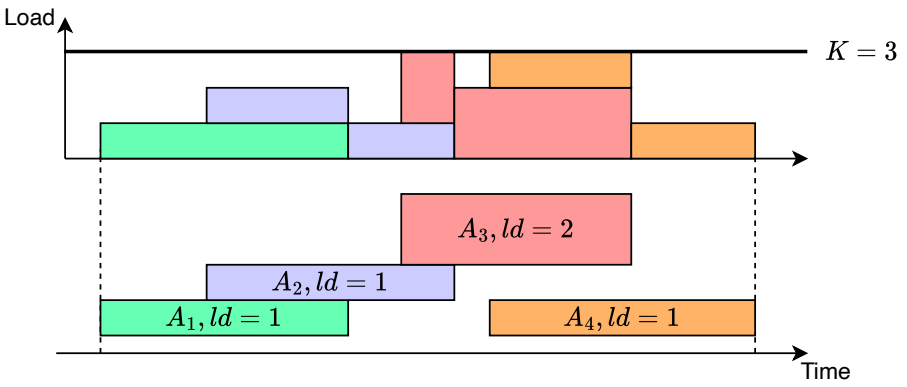

Figure 2.7: Resource consumption profile with four tasks for a capacity K of 3

2.2.2 Search

The search consists in assigning values to variables of the problem until a solution is found i.e. each variable is assigned to a single value. The constraints are used during the search to propagate changes in the domain of a variable to other variables thus restricting the search space. Conceptually, the search process can be described using a tree structure called a **search tree**. The root of the tree corresponds to the state at the start of the search. Each node is a partial assignment which differs from its parent by a single extension step. Leaves of the tree correspond to solutions where each variable is assigned.

This tree is generated and explored in a depth first fashion. At each node, the current state is expanded by **branching**. It consists in choosing a variable and applying a change to its domain. This triggers a propagation which leads to a new state where the process is repeated. As the domain of some variables is reduced trough propagation, it removes some possible branches of the search tree, thus reducing the sub-tree to explore. This process is called **pruning**. If an inconsistency is detected during the propagation or when a solution has been reached, the state is reverted to its parent node by **backtracking** and another branching decision is explored. The algorithm continues until the tree has been fully explored or if a resource limit has been reached. The pseudo-code for the search procedure is given in Algorithm 1.

Example 2.2.7. Let us consider a simple CSP with three variables: $x = \{1, 2\}, y = \{1, 2, 3\}, z = \{2, 3, 4\}$ and the constraints: $x \neq y \neq z$ and $x + y = z$. The search tree is given in Figure 2.8. The domains are indicated for each node. The crossed values correspond to values removed by propagation. The decisions are indicated on the edges. The blue arrows indicate in which order the tree is explored. Solution nodes are marked in green, inconsistencies in

Algorithm 1: Recursive CP search

```
   // recursive branching procedure:
 1 def branch(decision)
 2     apply decision ;
 3     propagate ;
 4     if inconsistency then
 5         return ;
 6     if all variables assigned then
 7         solution found ;
 8         return ;
 9     alts ← get branching alternatives ;
10     foreach decision ∈ alts do
11         branch(decision) ;
12     return ;
   // first call at the root of the search tree:
13 alts ← get branching alternatives ;
14 foreach decision ∈ alts do
15     branch(decision)
```

red. For the sake of the demonstration we will assume that the constraint $x + y = z$ propagates only when x or y is fixed. Notice that in this case, the branch $y \neq 1$ leads to an inconsistency. Otherwise, the value 4 would have been removed from the domain of z after the removal of 2 from y. This would have immediately led to the removal of 3 from the domain of y and the solution $x = 2, y = 1, z = 3$ would have been found without needing to branch on $y = 1$

2.2.2.1 Branching

Branching is the step of expanding the search tree. It is done by applying a change to the domain of one or more variables. Several possible changes are considered, each corresponding to a new branch in the search tree. For example, one could consider whether assigning a variable to a specific value or removing the value from the domain of the variables leading to a binary decision. This kind of branching is called **binary branching**. It is used in Example 2.2.7. However, branching may involve more than two branches such as considering all possible values for a variable or even combinations of variables to change. The search heuristic is responsible for selecting the changes to consider and the order in which the resulting branches are explored.

Once the branching decision is done, the change is applied and affected

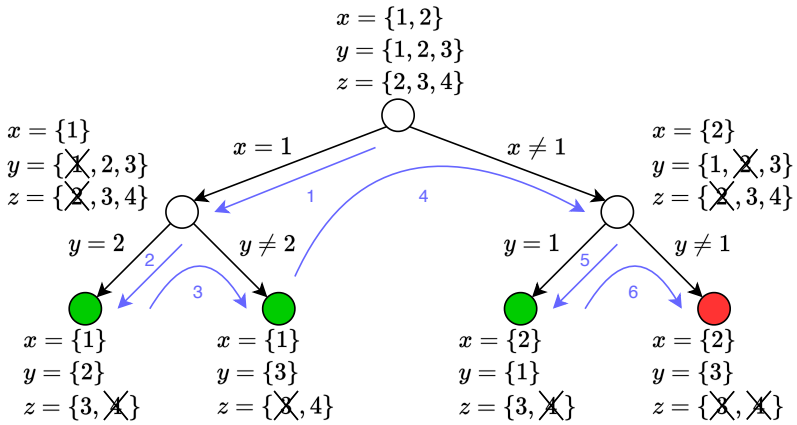

Figure 2.8: Example of CP search

constraints are notified which triggers propagation. The propagation continues until a fixed point is reached or an inconsitency is detected. In case of inconsistency or if all variables are assigned and a solution has been found, the domains are backtracked to their state before the branching decision and another branch is explored. Otherwise, a new branching step occurs.

2.2.2.2 Heuristic

A search heuristic guides the branching decisions during the search. At each branching step, it selects the possible changes to apply to the domain and the resulting branch to explore next. As we use a DFS search to explore the search tree, the decision to explore first a branch over another can heavily impact the search. Indeed, a branching decision leads to a domain reduction followed by a propagation, thus affecting the size of the resulting search space. Alternative branches may lead to sub-trees that can vary widely in side. Hence, selecting good branching decisions may greatly reduce the size of the search tree and thus its exploration speed. Heuristics can also guide the search towards better solutions which also speeds up the search for an optimal solution. Branching heuristics are thus a crucial component to consider in order to improve the search speed.

The general intuition of a search heuristic is that we want to select domain changes that are more likely to yield a large amount of propagation or a better solution. Search heuristics are usually separated in two categories: **variable** heuristics select the next variable to branch on while **value** heuristics select the next value to assign or remove from the domain of the selected variable. Common variable heuristics include:

- the **FirstFail** heuristics consists in selecting first variables that have a

higher chance to lead to failures. This usually leads to a better propagation during the search and might avoid some backtracks. Typically, variables with a smaller domain are selected first as they are more likely to be impacted by constraints.

- the **Conflict Ordering** heuristic [Gay+15] expands the idea of the first fail by ordering dynamically the variables such that the ones having led to the most recent conflicts have priority.

- another variant of the first fail approach is the **Weighted Degree** heuristic [Bou+04]. Each variable is associated a weight that reflects the amount of time that a constraint involving the variable has failed. This weight is updated during the search.

Many more variable heuristics are possible besides the generic ones described above. Generally, the heuristic is designed specifically for the problem and might exploit additional information over the variables and what they represent in the model. Value heuristics follow the same principle. For example, for a maximization problem, it would make sense to try first the largest values in the domain as they are more likely to lead to a good solution.

2.2.2.3 Backtracking

Backtracking occurs when an inconsistency has been detected, if a solution has been found or if all branching alternatives at a node of the search tree have been explored. It consists in reverting the domain of all the variables to its previous state in the parent node. Two alternative strategies can be used for restoring the state. The first one, called **copying**, consists in placing in memory the current state of the domain at each node of the search tree in order to be able to restore it later. The alternative, called **trailing**, is to use reversible data structures that allow to easily reverse their state to a previous one. Copying is used in the Gecode solver [Rei+09; Sch+] while trailing is used in OscaR [Osc12] and MiniCP [MSV21]. More details on the differences between copying and trailing can be found in [Sch99; CHN01; Kot10].

As the Oscar solver was used to develop the work in this thesis, the structures and CP components presented are based on trailing.

Sparse Set A sparse set [de +13] is a data structure that is often used to implement domains in a trailing environment. It consists in two arrays: `dense` which stores the actual elements in the sparse set and `sparse` which indicates the index at which an element is in `dense`. Elements are identified by indices. An integer `n` indicates the index of `dense` after which elements are considered removed.

- To access an element e, we look the value at the index of `sparse` corresponding to the element: `sparse`[e]. Then, we access the index of `dense` corresponding to this value to retrieve the element: `dense`[`sparse`[e]].
- To remove an element e from the sparse set, we swap the element with the one before n in `dense` while updating their indices in `sparse` to keep track of the elements. Then, we decrement n by 1.
- When used in a trailing context, the successive values of n are saved. To restore the sparse set to a previous state, we simply revert n to its value at the desired state. All the elements that had been removed since this state are situated between the two indices in `dense` and thus considered again as part of the sparse set.

These operations are illustrated in Figure 2.9. Note that using a sparse set in this way allows only successive removal operations (or insertions if we consider the set initially empty and all the elements after n inserted) but not to mix both. Thus, if a domain that both grows and decreases is needed, another data structure will be used.

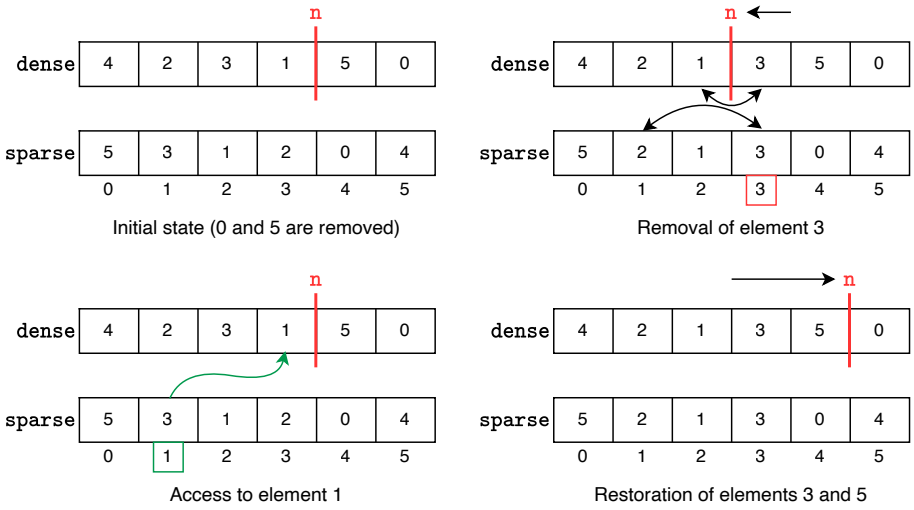

Figure 2.9: An illustration of the basic operations on a sparse set

2.2.2.4 Branch and Bound

For optimization problems, in addition to the branching strategy described in the previous sections, it is possible to use upper and lower bounds to improve the search speed. This paradigm is called **branch and bound**. The

principle is to maintain an upper bound UB during the search (in the case of a minimization problem). It is initialized to either ∞ or an overestimation of the optimal solution and tightened during the search, when new solutions are found. Before exploring a new branch b of the search tree, a lower bound for this branch $LB(b)$ is computed. If $LB(p) \geq UB$, meaning that no better solution can be found in this branch, the branch is pruned and not explored.

Another way to express this in the CP framework, is to add a constraint $O < O(s^*)$ each time a new improving solution s^* is found (in the case of a minimization problem, otherwise the constraint is $O > O(s^*)$).

2.2.2.5 Large Neighborhood Search

The Large Neighborhood Search (LNS) is a metaheuristic proposed by Shaw in [Sha98]. It aims at diversifying the solutions found by exploring different parts of the search tree. This is done by iteratively applying a partial relaxation of the current solution followed by a reconstruction in order to gradually improve the solution to the problem. The relaxation (also referred as a destroy method) consists in keeping a part of the current solution while leaving the remaining variables to their initial domain. This allows to restrict the search tree to the neighborhood of the current solution which is then explored in the hope of finding a better solution. A relaxation heuristic decides which parts of the current solution will be kept. It typically involves some part of randomness in order to relax different parts of the solution at each iteration and avoid cycling.

In CP, the relaxation imposes constraints that restrict some variables to their values in the current best solution. Then, the reconstruction (or search) heuristic guides the search in the resulting search space by assigning values to the remaining variables in order to find one or more new solution(s). Conceptually, LNS can be seen as jumping from one part of the search tree to another rather than exploring it in a complete fashion. The drawback of this approach is that the search is no longer complete. Thus, it is mainly used in contexts where the search tree is to large to be explored in its entirety.

Example 2.2.8. For example, a *random* relaxation heuristic selects randomly a percentage of the variables to relax and fix the other ones to their assignment in the current best solution. This heuristic can be parametrized by choosing the percentage to relax in a set of values such as $\{10\%, 20\%, 50\%\}$. A *first fail* heuristic with a fixed limit on the number of backtracks can be used as a reconstruction heuristic which can also be parametrized by choosing a limit on the number of backtracks in a set of values such as $\{50bkts, 500bkts, 5000bkts\}$.

The relaxation and reconstruction process continues until some limit in terms of iterations or time is reached. From a local search point of view,

CP is thus used as a slave technology for exploring a (large) neighborhood around the current best solution. LNS has been successfully used on various types of problems: bin-packing [Mal+13; Sch+11a], vehicle-routing [JV11; BH06; GD19], scheduling [GLN05; CB09; PH11; GSD14], traffic engineering [Har+15b], etc. Designing good relaxation and reconstruction heuristics with the right parameters is crucial for the efficiency of LNS. Unfortunately, this task requires some experience and intuition on the problem to solve. Nevertheless, there exists some advances in improving the genericity of LNS through some form of automatic subproblem selection [MSD10; PLJ14; Hen22]. Among them, the Adaptive LNS (ALNS) is an approach based on a portfolio of relaxation and search heuristics. It is discussed in details in Chapter 3.

2.2.3 CP Solvers

A constraint programming solver is a framework or software that implements a CP search engine and modeling layer. The user models its problem using the provided variables and constraints then uses the solver to search for solutions. Various CP solvers exist in diverse programming languages. Some are dedicated only to CP while others may also be bundled with support to use other search paradigms such as SAT or MIP. While most solver implement basic CP components, some more specific variables and constraints may be only implemented in a few of them. The solvers that were used or are discussed in this thesis are described in this section.

OscaR OscaR [Osc12] is a Scala based solver developed to solve a diverse array of Operations Research problems. It implements different techniques, among which a CP framework. OscaR is developed and maintained by a consortium of institutions and companies that includes part of the UCLouvain's AIA research group under the direction of Pierre Schaus. It is used for both academic research and commercial purposes. Most of the work done in this thesis was implemented and tested using OscaR.

MiniCP MiniCP [MSV21] is a recently developed lightweight solver that aims at teaching CP for newcomers in the field. It is implemented in Java and provides an extensive tutorial for the user to familiarize themselves with the solver and the underlying CP concepts.

CP Optimizer IBM ILOG CP Optimizer [Lab+18] is a commercial CP solver developed by IBM and mainly oriented towards scheduling problems. It provides advanced features such as dedicated variables and a powerful adaptive

search. Many of the concepts explored in this thesis have been implemented in some form in CP Optimizer, making it a useful point of comparison.

Google OR-Tools Google OR-Tools [PF19a] is an open source optimization suite maintained by Google that includes a CP solver. Like CP Optimizer, it provides a range of dedicated modeling and search features oriented towards routing and scheduling problems.

Modeling Languages Finally, in addition to CP solvers there exists constraint modeling languages. These languages are used to model CSPs and COPs at a high level, independently of a solver. The model can then be turned into a solver specific model to be run on a given solver. The advantage of such languages is that, as they are supported by a large range of solvers, they provide a generic way to model a problem. Two well known modeling languages are MiniZinc [Mina] (documentation is available at [Minb]) and XCSP [XCSa].

Adaptive Large Neighborhood Search

3

The first part of the work done in this thesis was spent on investigating generic resolution methods for CSPs and COPs. As the aim of the PRESupply project was to provide small and medium businesses with affordable solutions to tackle common optimization problems, one of the research directions explored was to devise generic and adaptive tools. Indeed, designing a dedicated solution for a specific problem is often costly and time consuming and thus often unaffordable for smaller businesses. Instead, once a generic solver has been developed to solve a range of common optimization problems, it can be quickly and easily used by businesses that encounter these problems.

In this context, the aim is to make the CP solver as automated as possible, allowing the user to quickly specify a model without spending time worrying about propagation or search. In order to make this goal possible, the idea explored in this chapter is to use an adaptive search method in a black-box fashion. The goal is thus to make search able to tune itself to the specificities of the problem without input from the user.

Back in 2004, Puget [Pug04] said that CP technology was too complex to use and more research efforts should be devoted to make it accessible to a broader audience. A lot of research effort has been invested to make this vision become true. Efficient black-box complete search methods have been designed [Ref04; HS17; Gay+15; CS15; MV12; PQZ12; VLS15] and techniques such as the embarrassingly parallel search are able to select the best search strategy with almost no overhead [PRS16]. For CP, Puget argued that the model-and-run approach should become the target to reach. The improvements went even beyond that vision since for some applications, the model can be automatically derived from the data [Pic+17; BS12].

This work aims at automating the CP technology in the context of Large Neighborhood Search (LNS) described in Section 2.2.2.5. In order to design an automated LNS, two approaches can be envisioned. A first one would be to recognize the structure of the model in order to select the most suited heuristic from a taxonomy of heuristics described in the literature. This approach, which is used in [MDV09] for scheduling problems, has two disadvantages:

1. some problems are hybrids and thus difficult to classify or recognize,

2. it requires a lot of effort and engineering to develop the problem inspector and to maintain the taxonomy of operators.

Therefore, we follow a different approach called Adaptive Large Neighborhood Search (ALNS) introduced in [RP06] which uses a portfolio of heuristics and dynamically learns on the instance which ones are the most suitable. At each iteration, a pair of relaxation and reconstruction heuristics is selected and applied on the current best solution. The challenge is to select the pair having the greatest gradient of the objective function over time (evaluated on the current best solution) based solely on the past executions.

We expand the usage of the Self Adaptive LNS (SA-LNS) framework proposed in [LG07] on different optimization problems by considering the model as a black-box. Our solver uses a set of generic preconfigured methods (operators) that hypothesize specificities in the problem and leverage them in order to efficiently perform LNS iterations. Given that the operators available in the portfolio are well diversified, we hope to provide a simple to use yet efficient framework able to solve a broad range of discrete optimization problems.

Our contributions to the ALNS framework are:

1. An adaptation of the weight update mechanism able to better cope with unequal running times of the operators.

2. A portfolio of operators easy to integrate and implement in any solver for solving a broad range of problems.

3.1 Related Work

While the ALNS framework has been used a lot on specific problems [PAM04; RP06; DBL12; Kov+12; RL12; MLP13; ACJ14; AGP14; Wen+16; SPR19], particularly in the field of vehicle routing, at our knowledge it has not yet been considered for a black box approach on multiple problems in constraint programming. Recent developments include the use of ALNS with Mixed Integer Programming (MIP) [Hen22]. Other portfolio based black box optimization approaches exist [BP14; He+19] but are mostly formulated as *multi-armed bandit* problems.

The Multi-Armed Bandit (MAB) problem [MT08; Sli19] consists in sequentially allocating resources between competing alternatives. Typically, this problem deals with the conflict between maximizing current profits versus investing in the hope of obtaining better future rewards. A portfolio based approach can be formulated as à MAB in the following way: at each iteration, a heuristic must be selected from the portfolio; the decision problem consists

in deciding between selecting the current best heuristic or trying another alternative in order to know if it is efficient on the problem at hand. Several algorithms have been proposed to tackle this problem [KP14] and it has already been considered in the context of constraint programming [Lot+13].

Another closely related field is the *reinforcement learning* [SB18]. It consists in using Machine Learning (ML) techniques in order to train agents at taking decisions by trial and error. The technique is based on a scoring function that rewards positive actions and punishes negative ones. This technique has recently been of interest in the domains of search heuristic selection [Nar03] and CP [MDV11; BFW11; Ant+20; Cha+21; Cap+21; LCP22].

3.2 Adaptive Large Neighborhood Search

Each ALNS operator as well as its possible parameters is associated to a weight. These weights allow to dynamically reward or penalize the operators and their parameters along the iterations to bias the operator selection strategy. Algorithm 2 describes the pseudo-code for an ALNS search. $\Delta c \geq 0$ is the objective improvement and Δt is the time taken by the operator.

Algorithm 2: Adaptive Large Neighbourhood Search for a minimization problem

```
1  s* ← feasible solution ;
2  do
3  |  relax ← select relaxation operator ;
4  |  search ← select search operator ;
5  |  (s', Δt) ← search(relax(s*)) ;
6  |  Δc ← cost(s*) − cost(s') ;
7  |  weight_relax ← updateWeight(relax) ;
8  |  weight_search ← updateWeight(search) ;
9  |  if Δc > 0 then
10 |  |  s* ← s' ;
11 while stop criterion met;
12 return s* ;
```

3.2.1 Roulette Wheel selection

We use the Roulette Wheel selection mechanism as in [LG07; PR07]. It consists in selecting the operators with probabilities proportional to their weight. The probability $P(i)$ of selecting the i-th operator o_i with a weight w_i among

the set of all operators O is

$$P(i) = \frac{w_i}{\sum_{k=1}^{|O|} w_k} \tag{3.1}$$

3.2.2 Weight evaluation

In [LG07], the authors evaluate the operators ran at each iteration using an efficiency ratio r defined as:

$$r = \frac{\Delta c}{\Delta t} \tag{3.2}$$

This ratio is then balanced with the previous weight of the operator $w_{o,p}$ using a reaction factor $\alpha \in [0, 1]$:

$$w_o = (1 - \alpha) \cdot w_{o,p} + \alpha \cdot r \tag{3.3}$$

While the reaction factor is important to accommodate the evolving efficiency of the operators during the search, this method does not cope well with operators having different running times. Indeed, operators with a small execution time will evolve faster as they will be evaluated more often. This can lead less efficient operators to be temporally considered better as their weight will decrease slower.

Example 3.2.1. Let us consider two operators A and B with running times of respectively 2 and 4 seconds. Both operators start with an efficiency ratio of 10 but after some time in the search, A has an efficiency of $\frac{1}{2}$ and B of $(\frac{1}{4})$. If each operator is separately run for 4 seconds, under a reaction factor of $\alpha = 0.9$; as A will be evaluated twice, its weight will decrease to 0.595 $(0.1 \cdot (0.1 \cdot 10 + 0.9 \cdot \frac{1}{2}) + 0.9 \cdot \frac{1}{2})$. Over the same duration B would be evaluated once and its weight would become 1.225 $(0.1 \cdot 10 + 0.9 \cdot \frac{1}{4})$. While both operators will eventually converge towards their respective efficiency, for a short amount of time, B will have a higher score than A and thus a higher probability to be selected.

This induces a lack of reactivity in the operator selection. In the following, we propose a variation of the weight update rule, more aligned with the expected behavior in case of different runtimes among the operators.

3.2.3 Evaluation window

The new mechanism proposed evaluates an operator based on its performances obtained in a sliding evaluation window:

$$[t^* - w, now] \tag{3.4}$$

where t^* is the time at which the last best solution was found and w is the window size meta-parameter. The window thus adapts itself in case of stagnation to always include a fixed part of the search before the last solution was found. This ensures that the operator(s) responsible for finding the last solution(s) will not have their score evaluated to 0 after a while in case of stagnation.

For each LNS iteration i, we record the operator used o_i, the time t_i at which it was executed, the difference Δc_i of the objective and the duration of execution Δt_i. We define the local/total efficiency ratio $L(o)/T(o)$ of an operator and the local/total efficiency L/T of all the operators as:

$$L(o) = \frac{\sum_{i|o_i=o \wedge t_i \in [t^*-w,now]} \Delta c_i}{\sum_{i|o_i=o \wedge t_i \in [t^*-w,now]} \Delta t_i} \tag{3.5}$$

$$T(o) = \frac{\sum_{i|o_i=o \wedge t_i \in [0,now]} \Delta c_i}{\sum_{i|o_i=o \wedge t_i \in [0,now]} \Delta t_i} \tag{3.6}$$

$$L = \frac{\sum_{i|t_i \in [t^*-w,now]} \Delta c_i}{\sum_{i|t_i \in [t^*-w,now]} \Delta t_i} \tag{3.7}$$

$$T = \frac{\sum_{i|t_i \in [0,now]} \Delta c_i}{\sum_{i|t_i \in [0,now]} \Delta t_i} \tag{3.8}$$

Intuitively, the local efficiency corresponds to estimating the gradient of the objective function with respect to the operator inside the evaluation window. If the operator was not selected during the window, its local efficiency is 0 which might be a pessimistic estimate. Therefore we propose to smooth the estimate by taking into account $T(o)$ normalized by the current context ratio L/T. The evaluation of an operator o is computed as:

$$weight(o) = (1-\lambda) \cdot L(o) + \lambda \cdot \frac{L}{T} \cdot T(o) \tag{3.9}$$

with $\lambda \in [0,1]$ a balance factor between the two terms. As we desire to evaluate the operator mainly based on its local efficiency, we recommend that $\lambda < 0.5$.

Note that initially, the weight of the operators is initialized to the same value and thus the first selection will be random. With some prior knowledge of the problem or about the operators genericity and average efficiency, it would be possible to bias this initial selection to favor operators that are expected to perform better. Finally, it is also possible to consider several variants of a same operator with different parameter values such as an allocated resource budget or a relaxation size. One could even implement some kind of nested selection by having a specific roulette wheel tied to each operator to select its parameter values.

3.3 Operator portfolio

In this section, we present the relaxation and search operators that we propose to be part of the portfolio. All of them operate on a vector of integer decision variables. This list is based on our experience and the features available in the solver used for our experiments. Therefore it should not be considered as exhaustive.

3.3.1 Relaxation Heuristics

- **Random** Relaxes randomly k variables by fixing the other ones to their value in the current best solution. This heuristic brings a good diversification and was demonstrated to be good despite its simplicity [LS14].
- **Sequential** Relaxes randomly n sequences of k consecutive variables in the vector of decision variables. This heuristic should be efficient on problems where successive decision variables are related to each other, for instance in Lot Sizing Problems [Fle90; Hou+14].
- **Propagation Guided and Reversed Propagation Guided** Those heuristics are described in [PSF04]. They consist of exploiting the amount of propagation induced when fixing a variable to consider together sets of variables whose values are strongly dependent on each other. To do so, the domain size of the variables is examined before and after the assignment of a variable. The difference of domain size allows to detect which variables are liked to the assigned variable. The basic propagation guided heuristic freezes together variables that are closely linked while the reverse variant ensures that such groups of variables are relaxed together.
- **Value Guided** This heuristic uses the values assigned to the variables. We have five different variants:
  - **Random Groups** Relaxes together groups of variables having the same value. This variant should be efficient on problems where values represent resources shared between variable such as bin-packing problems [SD+08].
  - **Max Groups** This variant relaxes the largest groups of variables having the same values. It can be useful for problems such as the Base Station Association and Power Control problem (BAPC) or Assembly line balancing [Mon+07].
  - **Min Groups** This method relaxes the smallest groups of variables having the same value (which can be single variables).

  - **Max Values** It consists in relaxing the k variables having the maximum values. We expect this heuristic to be efficient with problems involving a makespan minimization such as the job-shop problem.
  - **Min Values** This heuristic relaxes the k variables having the minimum values. It should be efficient in case of maximization problems.

  Note that another possible value selection heuristic that was not investigated would be to relax together variables with close values or which values are present in some interval. Such heuristic could be useful for scheduling problems.

- **K-Opt** This heuristic makes the hypothesis that the decision variables form a predecessor/successor model (where variable values indicate the next or previous element in a circuit). It is inspired by the k-opt moves used in local search methods for routing and clique problems [Uld+90; KHN04; KSN07]. The principle is to relax k edges in the circuit by selecting k variables randomly. The remaining variables have their domain restricted to only their successor and their predecessor in the current best solution in order to allow inversions of the circuit fragments. Figure 3.1 illustrates this process. The left graph shows the current solution as well as the edges that are cut by the relaxation. The right graph shows the resulting possible edges. Circuit fragments are colored in green while the other edges are colored in red. The arrays show the domains before (top) and after (bottom) the relaxation.

- **Precedence Based** This relaxation is useful for scheduling problems and hypothesizes that the decision variables corresponds to starting times of activities. It imposes a partial random order schedule as introduced in [GLN05]. To apply this technique in a black box context, the heuristic assumes that variables represent time values. k variables are selected to be relaxed. For those their domain is set free. The other variables have a partial order imposed based on their value in the current solution. To do so, variables are grouped and sorted by value. Precedence constraints are then added between variables in adjacent groups.

- **Cost Impact** This operator was described in [LS14]. It expands the principle of the propagation guided heuristic by considering the impact of the variable assignation on the objective function. To do so, the effect of the variables on the objective function is estimated through a cost metric that is measured through several dives. These dives consist in assigning the values of the current solution successively in a random

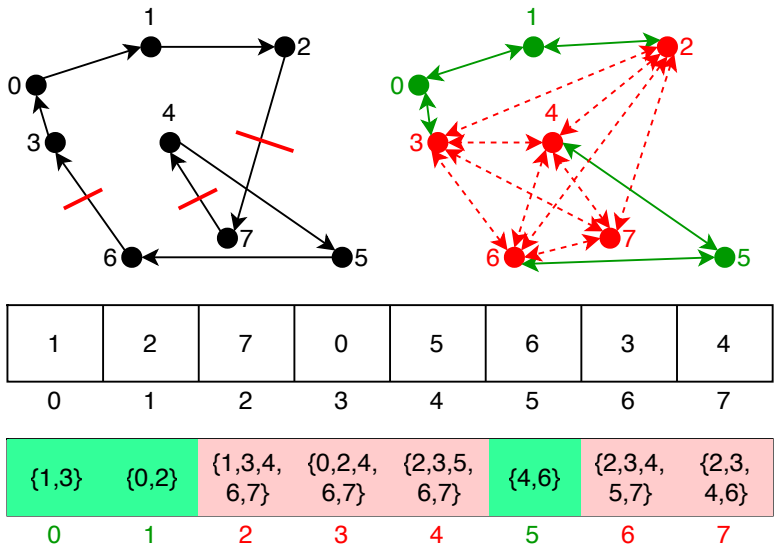

Figure 3.1: Illustration of the K-Opt relaxation

order. The difference of the objective domain size before and after the assignment of a variable is measured and added to the impact metric. This metric is then used as weight for a randomized selection, thus biasing the selection towards variables that impact the objective function the most.

Note that this list of operators is not exhaustive. Many other approaches could be considered such as using the constraint network to detect linked variables.

3.3.2 Search Heuristics

A search heuristic explores the search space of the remaining unbounded variables by iteratively selecting a variable and one of its values to branch on. They can be separated into two components: a variable heuristic and a value heuristic. Here are the variable heuristics used:

- **FirstFail** tries first variables that have the most chances to lead to failures in order to maximize propagation during the search.
- **Conflict Ordering** proposed in [Gay+15] reorders dynamically the variables to select first the ones having led to the most recent conflicts.
- **Weighted Degree** introduced in [Bou+04] associates a weight to each variable. This weight is increased each time a constraint involving that variable fails.

In combination with these variable heuristics, we used different value heuristics which select the minimum/maximum/median/random value in the domain in addition to the **value sticking** [FD94] heuristic which remembers the last successful assigned values. We also permit a binary split of the domain into $\leq, >$ branching decisions.

3.4 Experimental Results

As in [PRS16] we use an oracle baseline to compare with ALNS. Our baseline consists of a standard LNS with for each instance the best combination of operators (the one that reached the best objective value in the allocated time), chosen a posteriori. Notice that this baseline oracle is not the best theoretical strategy since it sticks with the same operator for all the iterations.

We implemented our framework in the OscaR constraint programming solver [Osc12] where it is available in open-source. We used the portfolio of operators presented in Section 3.3. We tested our framework on 10 different constraint optimization problems with two arbitrarily chosen medium-sized instances per problem. The problems are:

- The **Job-Shop** scheduling problem [Law84].
- The **Quadratic Assignment Problem (QAP)** [CB89].
- The **Resource-Constrained Project Scheduling Problem (RCPSP)** [SKD95].
- The **Steel Mill Slab** problem [Mig].
- The **Travelling Salesman Problem (TSP)** [KFM71].
- The **Vehicle Routing Problem with Time Windows (VRPTW)** [Sol87].
- The **Cutting Stock (Cutstock)** problem [GW95], instances from [XCSb].
- The **Graph Colouring** problem [Lew16], instances from [XCSb].
- The **Pigment Sequencing Problem (PSP)** [Hou+14].
- The **Incapacitated Warehouse Location Problem (UWLP)** [MV04], instances from [XCSb].

We compare:

1. An implementation of the approach from [LG07] (denoted *Laborie* here after) with a reaction factor α of 0.9.

2. The variant of [LG07] proposed in this thesis (denoted *Eval window*) with a sliding window $w = 10$ seconds and a balance factor λ of 0.05.

3. The oracle baseline.

Each approach was tested from the same initial solution (found for each instance using a first-fail, min-dom heuristic) with the same set of operators. We used relaxation sizes of $\{10\%, 30\%, 70\%\}$ and backtracks limits of $\{50\,bkts,\ 500\,bkts,\ 5000\,bkts\}$. We generated a different operator for each parameter(s) value(s) combination but kept the relaxation and reconstruction operators separated. We have 30 relaxation and 36 reconstruction operators, which yields a total of 1080 possible combinations to test for the baseline. Each ALNS variant was run 20 times with different random seeds on each instance for 240 seconds. We report our results in terms of cost values of the objective function for each instance. In order to compare the anytime behaviour of the approaches, we define the *relative distance* of an approach at a time t as the current distance from the best known objective (BKO) divided by the distance of the initial solution:

$$\frac{objective(t) - BKO)}{(objective(0) - BKO)} \tag{3.10}$$

A relative distance of 0 thus indicates that the best known solution has been reached.

We report the final results in Table 3.1. For each instance, we indicate the best known objective (BKO) and the results obtained after 240 seconds of LNS. For each approach, we report the average objective value (obj), the standard deviation (std) if applicable and the relative distance to the best known solution (rdist). The best results between the two evaluated approaches are highlighted in green. Figure 3.2 plots the average relative distance to the best known solution in function of the search time.

The results seem to indicate (at least on the tested instances) that the weight estimation based on an evaluation window tends to improve the performances of the original ALNS as described in [LG07]. The average relative distance to the best known solution is of 0.12 at the end of the search using the evaluation window, while it is of 0.18 using our implementation of [LG07]. None of the ALNS approaches is able to compete with the baseline (except on a few instances), but they obtain reasonably good solutions in a short amount of time. Furthermore, their any-time behavior is good when compared to the baseline and tends to get closer towards the end of the search.

Figure 3.3 shows a heat map of the relative selection frequency of the relaxation operators for each problem in the Eval window approach. The darker an entry, the more frequently this operator was selected for the problem instance. Two interesting observations can be made. First, a subset of

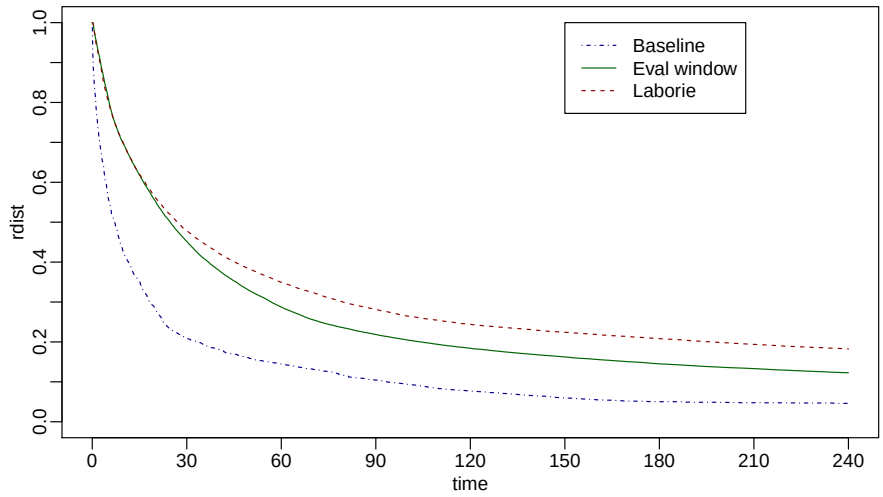

Figure 3.2: Average relative distance to BKO during the search

operators emerges more frequently for most of the problems. Second, this set varies between problems of different types, but is correlated between instances of the same problem. For some problems this set of operators is more uniform than others. For example, on the warehouse location and the cutting stock problems the operators are selected rather uniformly. The job shop has a strong preference for the max-val and precedency operators. On the contrary, cost-impact is almost useless for the makespan objective of the job shop. Not surprisingly the RCPSP, also a scheduling problem, selects the same two operators as the job shop. The random operator is generally good except for scheduling problems. These results confirm our intuition and a priori experience of which operator would be the most successful on each problem.

Table 3.1: Experimental results

Instance	Problem	BKO	Baseline		Eval window			Laborie		
			obj	rdist	obj	std	rdist	obj	std	rdist
la13	JobShop	1150	1150	0	1157.65	12.06	0	1195.65	156.4	0.01
la17	(Lawrence-84)	784	784	0	784.05	0.22	0	784	0	0
chr22b	QAP	6194	6292	0.01	6517	115.16	0.04	6626.6	135.71	0.06
chr25a	(Christofides-89)	3796	3874	0	4682	393.52	0.04	4982.3	342.88	0.06
j120_11_3	RCPSP	188	228	0.08	420.75	129.61	0.48	399.55	62.88	0.43
j120_7_10	(Kolisch-95)	111	374	0.49	160.6	113.23	0.09	127.7	1.42	0.03
bench_7_1	Steel	1	6	0.03	30.65	11.69	0.18	49.75	8.92	0.29
bench_7_4	(CSPLib)	1	9	0.04	24.3	5.83	0.12	42.05	8.05	0.21
kroA200	TSP	29368	31466	0.01	50022.35	11159.91	0.06	156239.10	12097.24	0.37
kroB150	(Krolak-72)	26130	26141	0	28596.2	872.77	0.01	61658.7	6051.92	0.14
C103	VRPTW	82811	82814	0	105876.1	5553.99	0.07	137749.3	17371.97	0.17
R105	(Solomon-87)	137711	137261	0	140162.3	1593.46	0.02	144273.05	2051.77	0.05
t09-4	Cutstock	73	73	0	76.65	3.38	0.1	77.1	1.73	0.11
t09-7	(XCSP)	161	161	0	161	0	0	161	0	0
qwhopt-o18-h120-1	Graph colouring	17	17	0	17	0	0	17	0	0
qwhopt-o30-h320-1	(XCSP)	30	30	0	541.4	51.9	0.59	653.2	32.46	0.72
PSP_100_4	Lot sizing	8999	9502	0.03	11796.85	1204.18	0.18	14682.45	847.30	0.36
PSP_150_3	(Houndji-2014)	14457	16275	0.23	18236.15	569.8	0.48	19482.2	339.11	0.63
cap101	Warehouse	804126	804126	0	804599.55	428.34	0	804556.5	430.5	0
cap131	(XCSP)	910553	910553	0	913147.6	2701.98	0	913240.4	3197.29	0
Average				0.05		1246.05	0.12		2156.88	0.18

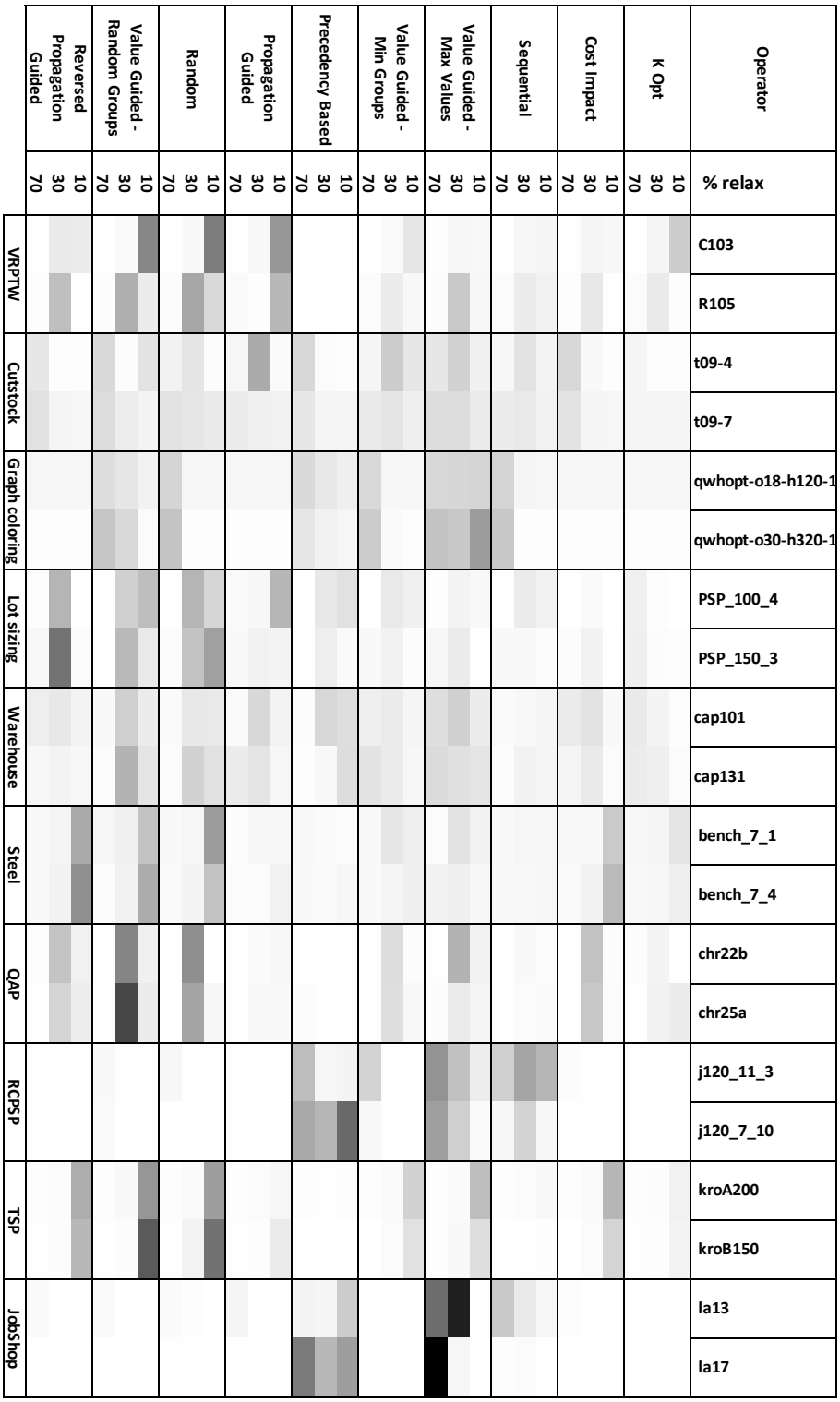

Figure 3.3: Heat map of the relaxation operators selection for the Eval window approach

3.5 Conclusion

This chapter studied the use of an Adaptive Large Neighborhood Search (ALNS) to solve problems in a black-box context. The weight update mechanism of the ALNS framework was studied and a portfolio of relaxation and search operators was proposed. The contributions of this chapter are:

- A new weight update mechanism that is based on an evaluation window and is able to better deal with operators presenting variations in running time (Section 3.2.3).
- A portfolio of relaxation and search operators designed to solve various optimization problems in a black-box context (Section 3.3). This portfolio includes original operators such as the K-Opt relaxation heuristic.
- An experimental evaluation of the ALNS framework on a set of 10 different optimization problems (Section 3.4).

The experimental results show that the evaluation window approach proposed improves the performances of the ALNS compared to the efficiency ratio from [LG07]. Additionally, the results highlighted by Figure 3.3 show that the ALNS is able to select efficient operators for the problem at hand among those present in the portfolio. This comforts us that self-adaptive LNS could reach the performances of an expert that would select the operators manually for each problem.

Patient Transportation Problem

4

After the development of the black-box search strategy presented in Chapter 3, the focus of this thesis was turned to the resolution of an optimisation problem provided by the Centrale de Services à Domicile (CSD) [CSD], one of the partners in the PRESupply project. The CSD is a non-profit organization operating at Liège (Belgium) which provides a range of home help services. One of them is transportation of people to medical appointments. The problem to solve, which is called the Patient Transportation Problem (PTP), is to route a fleet of vehicles in order to transport patients between their homes and medical appointments.

Over the years, there is an increasing demand for transports by disabled and invalid people requiring health care but that do not have the ability to go to hospitals by themselves. In this context, organizations managing the transportation of patients from their home to health centers are present in many cities. Their goal is to provide a door-to-door transportation service to a set of patients on a daily basis. Most of them are non-profit organizations that often have limited resources. Besides, they often do not have an expertise on decision support tools in order to assist them in their operations. This leads to sub-optimal decisions in most cases which has a direct negative impact on the patients and also leads to financial losses. Therefore, minimizing the operational costs while maintaining a sufficient quality of service is highly desirable and both aspects must be properly balanced.

The Patient Transportation Problem, is a specific case of the well-known Dial-a-Ride Problem (DARP) [CL07]. The goal of this last problem consists in designing routes and schedules for a set of users who specify pickup and delivery requests between origins and destinations. It is especially present for the transportation services in the medical domain [MM11; Liu+13; DPd17].

As a first observation, we can see that most of the approaches are based either on Mixed Integer Programming, Local Search or Dynamic Programming. Conversely, solutions based on Constraint Programming (CP) seem to have been less studied even if some recent works exist [BPR11; BCL12; PS13; JV11; LAB18]. However, thanks to its flexibility, we believe that CP can play an important role for solving practical DARPs.

4.1 Problem Description

The Patient Transportation Problem (PTP) is a static optimization problem aiming to bring patients to health centres and to take them back home once the care has been delivered. To do so, a fleet of vehicles is available. The fleet is heterogeneous and is mainly composed of ambulances and private drivers operating as volunteers. Each patient has a set of characteristics and is represented by a request. The objective is to satisfy as many requests as possible within a fixed horizon, which is typically bounded by the working hours. Three aspects of decision are considered in the PTP:

1. selecting which requests to service;
2. assigning vehicles to requests;
3. routing and scheduling appropriately the vehicles.

An illustration of the PTP on a toy example with two patients (A and B) and a single vehicle is shown in Figures 4.1 and 4.2. A possible solution consists in the following sequence: taking A (S_1), bringing A to the hospital (S_2), taking B (S_3), taking back A (S_4), dropping A to its home (S_5), bringing B to the hospital (S_6), waiting for B (S_7) and dropping B to its home (S_8).

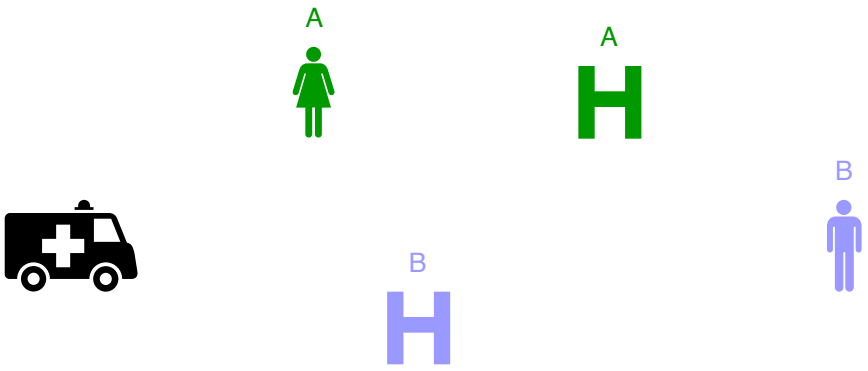

Figure 4.1: Illustration of the initial situation of a PTP with one vehicle and two patients

Some specific characteristics must also be considered in the PTP. Here are some of them:

- Patients can have several constraints such as a maximum travel time or a maximum waiting time at the hospital. The time to embark and disembark a patient must sometimes be considered and might differ between patients.

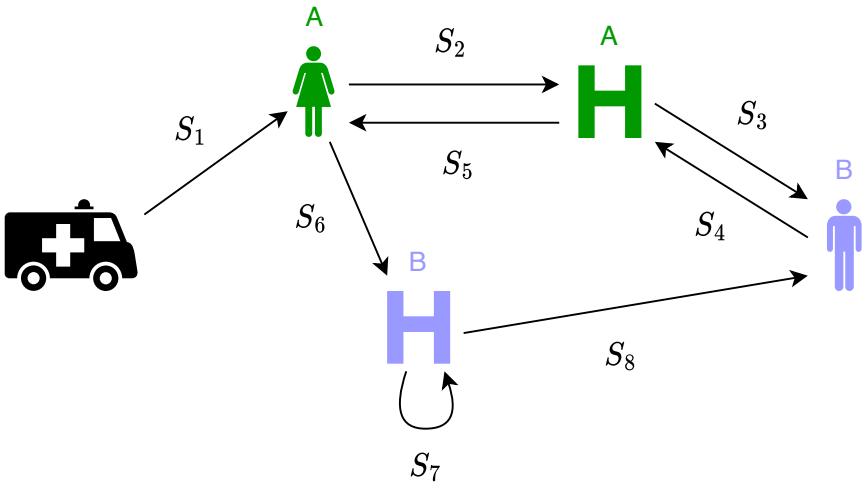

Figure 4.2: a possible solution for the PTP with one vehicle and two patients

- The set of requests is heterogeneous. Some patients only require to go from their home to a health center, while some of them also need a return trip once the care has been delivered. In the latter case, they must be taken back home, or to another place if requested. It is also possible to have patients requiring only a return trip. Besides, requests can involve more than one passenger at once. For instance, a child can be accompanied by his parents.

- The vehicle fleet is heterogeneous. Vehicles can differ by their capacity, their initial/final location (typically a depot) and their availability. Some patients can only be taken by particular vehicles. For instance, patients in wheelchairs can only be transported by specific vehicles.

- Availability of vehicles can be non continuous. For instance, they can be available from 9am to 1pm and from 3pm to 6pm.

Note that this version of PTP is static: the whole set of requests is known beforehand and no new request is added in real time. It is used by the organization for designing the first daily schedule given the pool of requests received the previous days. Let us finally notice that as a variant of the DARP, the PTP is a *NP*-complete problem [BKS98].

4.1.1 Formal definition

The version of the problem solved in this chapter is defined in formal terms as such: Similarly to the DARP, the PTP is described on a complete graph

$G = (X, E)$ and a transition matrix $trans_{i,j}$ contains the distance for each edge $(i, j) \in E$.

With each PTP request $r \in R$ is associated the service time required to embark or disembark the vehicle srv_r, the load or number of places required in the vehicle ld_r, the category of patient cat_r, the time of the appointment rdv_r, the duration of the care $drdv_r$, and a parameter $maxw_r$ that indicates the maximum waiting time for the request defined as the difference between the start (or end for backward travels) of the patient appointment and the departure from the origin stop (or arrival at the return stop for backward travels). Each request consists in one or two travels (in case of two-way trip).

The set of all travels is noted T. Each travel $t \in T$ is composed of a couple of stops (p, d) where $p \in X$ corresponds to the pickup of the patient and $d \in X$ to their drop. The notations $t \in r$ and $i \in r$ indicates that the travel t or stop i is part of the request r. A travel $t \in r$ is either forward: from a starting place $org_r \in X$ to a destination place $dst_r \in X$ or backward: from dst_r to a return place $ret_r \in X$. Note that a request can either include a single forward travel, a single backward travel or both a forward and a backward travels. In the two first cases, the drop that is not used (ret_r or org_r) has a value of -1.

PTP vehicles are heterogeneous. Each vehicle $v \in V$ has a capacity k_v, a set C_v of request categories it can serve, a start depot $sd_v \in X$, an end depot $ed_v \in X$ and a time window $[savail_v, eavail_v]$ in which the route must be done.

Each PTP stop $i \in X$ is associated to a time window $[ea_i, la_i]$: For forward travels, the time window of each stop $i \in r$ is $[rdv_r - maxw_r, rdv_r - srv_r]$; For backward travels the initial window is $[rdv_r + drdv_r, rdv_r + drdv_v + maxw_r - srv_r]$; For depot stops of a vehicle v the window is $[savail_v, eavail_v]$.

The essential parameters of the problem are summarized in Table 4.1. The rest of the parameters can be computed based on these essential parameters.

Note a few differences with the DARP problem defined in section 2.1.5: Rather than being defined by global parameters (X_0, $[0; Tmax]$ and K), the depots (sd and ed), availability windows ($savail$ and $eavail$) and capacities k can be different for each vehicle. There is no maximum route duration parameter D_{max}. The requests have a parameter $maxw$ that indicates the maximum wait time of the patient. The requests also include a parameter cat which is the category of the patient transported. Conversely, each vehicle is characterized by a set of patient categories C that it can take. Finally, the requests can include one or two travels. This last difference is important as it introduces dependencies between some travels.

The objective of the PTP is to maximize the number of requests serviced (0) under the following constraints:

(1) Each vehicle v begins at its depot sd_v;

Table 4.1: Essential parameters of the problem

Entity	Parameter	Meaning
Request	org_r	Starting place of the patient of request r.
	dst_r	Place where the care is delivered for the patient of request r.
	ret_r	Return place of the patient of request r.
	srv_r	Service time for request r.
	l_r	Number of places taken in a vehicle for request r.
	rdv_r	Time at which the health care service begins for request r.
	$drdv_r$	Time needed to deliver the care for the patient of request r.
	$maxw_r$	Maximum waiting time for request r.
	c_r	Category of patient of request r (wheelchair, without, etc.).
Vehicle	k_v	Capacity of vehicle v (i.e. the number of places available).
	C_v	Set of patient categories that vehicle v can take.
	sd_v	Starting depot for vehicle v.
	ed_v	Return depot for vehicle v.
	$savail_v$	Time at which vehicle v can start its route at its starting depot.
	$eavail_v$	Time before which vehicle v must end its route at its return depot.

(2) Each vehicle v ends at its depot ed_v;

(3) The load of any vehicle v never exceeds its capacity k_v;

(4) For each pair of stops serviced by a same vehicle (i, j), the difference between the arrival time at the second stop (j) and the departure at the first stop (i) is higher or equal to the travel distance between the two stops $trans_{i,j}$;

(5) For each stop i, the service of the stop starts inside its time window $[ea_i, la_i]$;

(6) A pickup stop p_t is always visited before the associated drop stop d_t;

(7) A pickup stop p_t and its associated drop stop d_t are both serviced by the same vehicle;

(8) For each stop i associated to a request r, the service time to embark or disembark the vehicle srv_r is respected;

(9) For each stop $i \in r$, the servicing vehicle v is compatible with the stop $(cat_r \in C_v)$;

(10) The route of each vehicle v is contained in its availability window $[savail_v, eavail_v]$;

(11) A patient wait time never exceeds its maximum wait time $maxw_r$.

Note that constraints (1) to (8) are essentially the same as in the DARP. Constraints (9), (10) and (11) from the DARP are replaced by different constraints.

4.2 Related Work

To the best of our knowledge, the approach of Liu et al. [LAB18] is the closest and most recent work related to our problem. The authors model and solve the Senior Transportation Problem (STP) using different approaches: CP, MIP and Logic Based Benders Decomposition. The objective is also to maximize the sum of the (weighted) served requests. Their results show that the CP model has the best performances. The STP shares many similarities with our problem but has nevertheless some differences:

- Requests are one-way only and there is no return trip.
- The problem is a transportation problem where the selection of each request is constrained only by the vehicles availability and a maximum travel time, there are no constraints related to the appointment for care.
- There are no constraints linking patients to specific vehicles.

While some constraints are straightforward to add in the STP model, the integration of others requires more modifications. For instance, by properly defining the time windows to make sure the patients arrive on time for their care, appointment constraints for the care can be handled by the STP. However, additional constraints are necessary to link forward with backward trips and preserve the consistency of the tour. Ensuring that vehicles are the same or can be different for both trips also requires some modifications.

Besides, the modeling and solving parts are also different. In the approach of Liu et al. [LAB18], each decision variable is linked to a location and auxiliary variables are introduced to express that a location is visited by a particular vehicle. In our model, the decision variables are linked to trips instead of visited locations. We use simple integer variables coupled with constraints to represent activities and time windows. In addition, we express capacity constraints with the standard `cumulative` constraint [BCR12] and can take advantage of efficient propagators [GHS15a; Vil11; GHS15b; OQ13; Sch+11b]. Conversely, Liu et al. use the dedicated variables of CP Optimizer to represent activities, time windows and vehicle routes as well as enforce the capacity constraints of vehicles through renewable resources and *cumul* functions using the `StepAtStart` functions from CP Optimizer. Those abstractions are less standard in CP solvers and modeling languages such as Minizinc [Net+07] or XCSP3 [Aud+20] (renewable resources can be modeled with cumulative constraints [SC95]). Finally, we use a custom search strategy combined with a Large Neighborhood Search while Liu et al. rather uses the CP Optimizer default search.

After the publication of the initial paper on the PTP that contains the other models presented in this chapter, the model for the STP proposed by

Liu et al. was adapted to the PTP. The implementation is done on the java API of the academic version of IBM ILOG CPLEX CP Optimizer V12.8. It is based on the CP model for the STP proposed in [LAB18] with the following modifications to accommodate the additional constraints of the PTP:

- The compatibility of vehicles with patients is dealt with by restricting the initial domain of the sequence variables representing the route of each vehicle to filter out auxiliary activities that represent stops of incompatible patients.
- The appointment time constraints are enforced using the dedicated support of CP Optimizer for time windows for activity variables.
- The forward and backward travels of the requests are bound together by using additional variables. Each travel activity is linked by a service boolean variable that indicates whether the activity is performed or not. The service variables corresponding to travels for a same request are linked together using an equality constraint. This ensures that both travels are either serviced if the request is taken or not serviced if the request is refused. Additionally, these variables are also linked to a dedicated service variable for each request which is in turn used in the objective function.
- The objective is changed to maximizing the number of serviced requests. This is done by binding the objective variable with the aforementioned request service variables using a sum constraint.

The search used is CP Optimizer default adaptive LNS search. The comparison between this approach and the others presented in this chapter is discussed in section 4.6.5.

4.3 Scheduling Model

The first approach considered is based on a modeling of the problem as a constrained based scheduling problem. The intuition is to represent the transportation of a patient as an activity performed by the vehicle. Doing so allows to use powerful constraints designed for scheduling problems such as the cumulative resource constraint [BCR12].

4.3.1 Decision Variables

The problem is modeled as a scheduling problem with conditional activities using the formalism proposed by Laborie et al [LR08; Lab+09; Lab+18]. Thus, each travel $t \in T$ will be modeled as a conditional activity A_i. In the standard

form, each conditional activity A_i is modeled with four variables, a start date $s(A_i)$, a duration $d(A_i)$, an end date $e(A_i)$ and a binary execution status $x(A_i)$. If the activity is executed, it behaves as a classical activity that is executed on its time interval, otherwise it is not considered by any constraint. In our case, we also define $v(A_i)$ as the vehicle that has been assigned to an activity A_i. Note that since the vehicle variable is linked to the activity, Constraint (7) is implicitly enforced. Each request r is attached to a forward activity (A_r^F) defining the time slot when the patient is brought from its home to the health center (from org_r to dst_r) and to a backward activity (A_r^B) for the time interval of the return trip (from dst_r to ret_r). Furthermore, A_i denotes any activity, either forward or backward, A^F the set of forward activities and A^B the set of backward activities. Equation 4.1 defines A_r^o and A_r^d as the origin and the destination locations of the activities linked to a request r.

$$\forall r \in R : \begin{cases} A_r^o = \begin{cases} org_r & \text{if } A_r \in A^F \\ dst_r & \text{if } A_r \in A^B \end{cases} \\ A_r^d = \begin{cases} dst_r & \text{if } A_r \in A^F \\ ret_r & \text{if } A_r \in A^B \end{cases} \end{cases} \tag{4.1}$$

Temporal relations between activities are illustrated in Figure 4.3 for an arbitrary example. The focus is on activity A_i^F. There are four specific transition times ($trans_{i,j}$) with any other activity (A_j^F on this example), they correspond to the time to go from A_i^o to A_j^o, from A_i^o to A_j^d, from A_i^d to A_j^o and from A_i^d to A_j^d. Activity A_i^F must also be completed before the appointment of the request (rdv_r), and the related backward activity cannot begin before the end of the appointment ($rdv_r + drdv_r$). Note that the service time srv_r is included in the corresponding activities (at the beginning and at the end) for each request $r \in R$ Finally, each activity is executed on a resource v, representing the vehicle assigned to the activity.

Decision variables related to the selection of requests are depicted in Equation (4.2). They are boolean variables defining whether the request is selected or not.

$$\forall r \in R : \; S(r) \in \{0, 1\} \tag{4.2}$$

Variables related to the conditional activities are shown in Equation (4.6) and illustrated in Figure 4.4. Patients cannot arrive at the health center after the time at which the appointment begins (forward activity) and cannot leave it before the end of the care (backward activity). The domain of the vehicle selection variables ($v(r)$) contains only the vehicles that are compatible with the patient category cat_r of the request (Constraint (9)). Domains for forward activities implicitly handle the deadline satisfaction for the care for each request. It ensures that the patients arrive to the health center ahead of

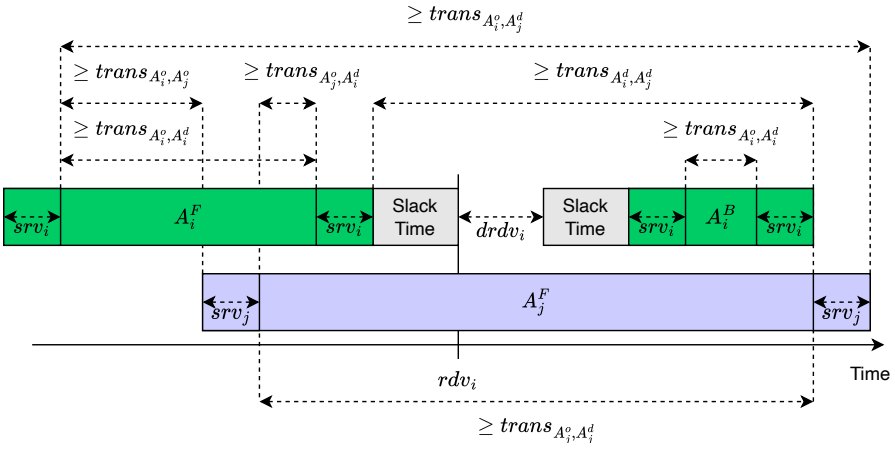

Figure 4.3: Temporal dependencies between activities

schedule for their care (Constraint (5)):

$$\forall r \in R: \; e(A_r^F) \leq rdv_r \tag{4.3}$$

Similarly, domains for backward activities ensure that patients cannot leave the center before the time at which the care has been delivered (Constraint (5)):

$$\forall r \in R: \; s(A_r^B) \geq rdv_r + drdv_r \tag{4.4}$$

The maximum waiting time (Constraint (11)) is also enforced through the initial domain of the time windows:

$$\forall r \in R: \; s(A_r^F) \geq rdv_r - maxw_r \wedge e(A_r^B) \leq rdv_r + drdv_r + maxw_r \tag{4.5}$$

$$\forall r \in R: \begin{cases} s(A_r^F) \in [rdv_r - maxw_r, rdv_r] \\ e(A_r^F) \in [rdv_r - maxw_r, rdv_r] \\ d(A_r^F) = e(A_r^F) - s(A_r^F) \\ x(A_r^F) \in \{0,1\} \\ v(A_r^F) \in \{j \mid j \in V \wedge c_r \in C_j\} \\ s(A_r^B) \in [rdv_r + drdv_r, rdv_r + drdv_r + maxw_r] \\ e(A_r^B) \in [rdv_r + drdv_r, rdv_r + drdv_r + maxw_r] \\ d(A_r^B) = e(A_r^B) - s(A_r^B) \\ x(A_r^B) \in \{0,1\} \\ v(A_r^B) \in \{v \mid v \in V \wedge cat_r \in C_v\} \end{cases} \tag{4.6}$$

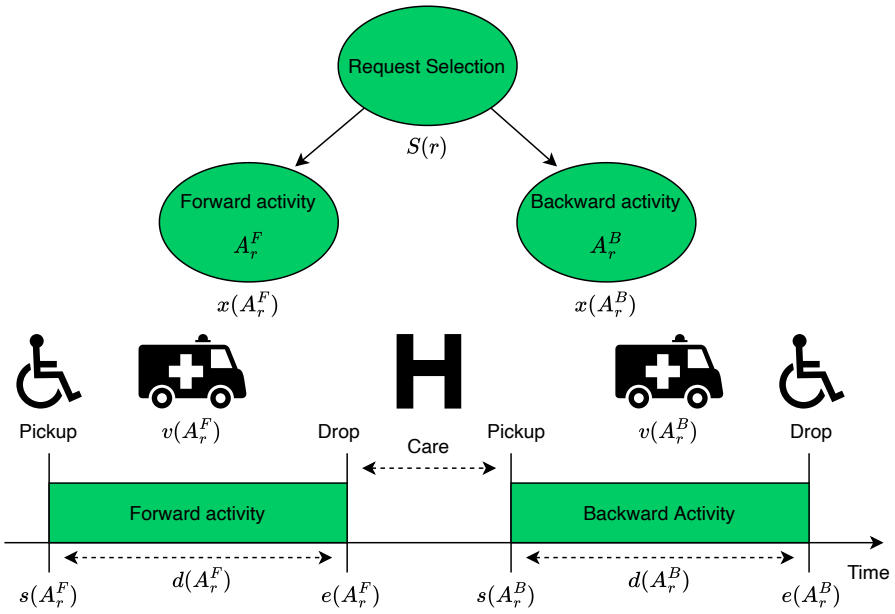

Figure 4.4: Illustration of the variables linked to a request r

4.3.2 Constraints

Binding Requests to Activities A request is selected if and only if the forward and backward activities are both completed:

$$\forall r \in R: \ \big(S(r) = 1\big) \equiv \big(x(A_r^F) = 1 \wedge x(A_r^B) = 1\big) \tag{4.7}$$

Forward and Backward Selection A forward and backward activity linked to the same request must have the same execution status (Equation (4.8)). This constraint is redundant with Equation 4.7 but can nevertheless be used for a better pruning.

$$\forall r \in R: \ x(A_r^F) = x(A_r^B) \tag{4.8}$$

Inter-Activity Time Travel Consistency The start/end of an activity cannot overlap with the start/end of other activities when they are processed by the same vehicle. The time interval between any two locations visited by a same vehicle is at least the time required to travel between these two locations (Constraint (4)). It is also referred as *setup time*. It is illustrated in Figure 4.3. Additionally, the service time to pickup or drop a patient must be taken into account (Constraint (8)). These constraints are enforced through transition constraints posted for each pair of activities (Equation (4.9)). The $\vee$ relation

is used to consider situations where activity A_i occurs before or after A_j.

$$\forall i, j \in R \mid i \neq j : \begin{cases} \big(v(A_i) = v(A_j)\big) & \rightarrow \begin{pmatrix} \big(s(A_j) - s(A_i) \geq trans_{A_i^o, A_j^o} + srv_i\big) \vee \\ \big(s(A_i) - s(A_j) \geq trans_{A_j^o, A_i^o} + srv_j\big) \end{pmatrix} \\ \big(v(A_i) = v(A_j)\big) & \rightarrow \begin{pmatrix} \big(s(A_j) - e(A_i) \geq trans_{A_i^o, A_j^d}\big) \vee \\ \big(s(A_i) - e(A_j) \geq trans_{A_j^o, A_i^d}\big) \end{pmatrix} \\ \big(v(A_i) = v(A_j)\big) & \rightarrow \begin{pmatrix} \big(e(A_j) - s(A_i) \geq trans_{A_i^d, A_j^o} + srv_i + srv_j\big) \vee \\ \big(e(A_i) - s(A_j) \geq trans_{A_j^d, A_i^o} + srv_j + srv_i\big) \end{pmatrix} \\ \big(v(A_i) = v(A_j)\big) & \rightarrow \begin{pmatrix} \big(e(A_j) - e(A_i) \geq trans_{A_i^d, A_j^d} + srv_j\big) \vee \\ \big(e(A_i) - e(A_j) \geq trans_{A_j^d, A_i^d} + srv_i\big) \end{pmatrix} \end{cases} \tag{4.9}$$

An alternative way to enforce the travel times is to use a *NoOverlap* (see Section 2.2.1.2) with transition time constraint imposed on activities created at each location [LAB18]. In particular, the propagator proposed by Dejemeppe et al. [DVS15; Van+16; VDS20] could possibly be extended to handle optional activities. But the decomposition approach relying on reification and binary constraints is arguably the most portable formulation for other solvers and modeling languages.

Intra-Activity Time Travel Consistency The duration of each activity cannot be lesser than the time required to go from the origin to the destination (Constraint (4)) (Equation (4.10)). Additionally, the service time to pickup and drop a patient must be taken into account (Constraint (8)). This constraint also enforces the precedence between the pickup p_t and the drop d_t for each travel t (Constraint (6)).

$$\forall r \in R : \; d(A_r) \geq trans_{A_r^o, A_r^d} + 2 * srv_r \tag{4.10}$$

Cumulative Resource At any moment, the number of places occupied by patients in a same vehicle v cannot exceed its capacity k_v (Constraint (3)):

$$\forall v \in V : \; cumulativeResource\Big(\Big\{(A_i, ld_i) \mid i \in T \wedge v(A_i) = v\Big\}, k_v\Big) \tag{4.11}$$

This constraint corresponds to the cumulative global constraint described in Section 2.2.1.2. In our case, each activity A_r consumes ld_r resources. We use the filtering algorithm of Gay et al. [GHS15a]. The vehicle of a non-executed activity is not considered by the constraint.

Vehicles Availability Vehicles also have constraints on their availability (Constraint (10)). They are available during a period and cannot leave their initial position (i.e. a depot) before the period (Constraint (1)). Similarly, they have to go back to their end depot before the end of the period (Constraint

(2)). These are characterized by the parameters sd_v the starting location of a vehicle v, ed_v its return depot and $[savail_v, eavail_v]$ its availability window. We define $D_r^o = sd_{v(A_r)}$ and $D_r^d = ed_{v(A_r)}$ as the origin/destination location of the vehicle linked to activity A_r as defined in Equation (4.6). Constraints on vehicle availability are expressed in Equation (4.12). It states that an activity cannot begin before the availability of its vehicle plus the time required to go from the initial depot to the patient place. Similarly, the vehicles must have enough time to return to their depot in order to stay in the availability window.

$$\forall r \in R : \begin{cases} s(A_r) \geq savail_{v(A_r)} + trans_{D_r^o, A_r^o} \\ e(A_r) \leq eavail_{v(A_r)} - trans_{A_r^d, D_r^d} \end{cases} \tag{4.12}$$

Empty Locations Some patients only require to go from their home to a health center without return trip. It is also possible to have patients needing only a trip from the health center to their home. A location can thus be empty which is indicated by a value of -1 in either an origin location org_r or a return location ret_r. When the start location is empty the request has no forward activity. Similarly, there is no backward activity when the return location is empty. In such cases, some constraints are simplified or not posted in order to consider only situations involving a single forward or a backward activity. More specifically, Equation (4.7) is adapted as shown in Equation (4.13) ($\vee$ instead of $\wedge$) and the constraint in Equation (4.8 does not hold anymore).

$$\forall r \in R : \ \big(S(r) = 1\big) \equiv \big(x(A_r^F) = 1 \vee x(A_r^B) = 1\big) \tag{4.13}$$

4.3.3 Objective Function

The criterion considered for the objective function is the satisfaction of requests (0). We want to maximize the number of served requests (Equation (4.14)).

$$\max\Big(\sum_{r \in R} S(r)\Big) \tag{4.14}$$

4.3.4 Extensions of the Model

One of the main asset of this model is its flexibility to easily accommodate new constraints depending on the situation. This section presents some variants of the problem and how they can be integrated in the core model.

Mandatory Requests It is possible to enforce the selection of some requests (Equation (4.15)). Parameter m_r is a boolean value indicating if a request r is mandatory.

$$\forall i \in \left\{ r \mid r \in R \wedge m_r = 1 \right\} : \; S(r) = 1 \tag{4.15}$$

Maximum Ride Time It is also possible to constraint the maximal ride time of patients. It prevents situations where a patient stays too long in a vehicle. To do so, the duration of each activity would be constrained based on a parameter $maxr_r$ (Equation (4.16)).

$$\forall r \in R : \; d(A_r) \leq maxr_r \tag{4.16}$$

Non Continuous Vehicle Availability Some vehicles can have non continuous availability. For instance, they could be available from 9am to 1pm and from 3pm to 6pm. We would handle this specificity by duplicating the vehicles for each continuous interval. The availability of each vehicle is then composed by a unique interval. In most practical cases, vehicles would be duplicated only once (morning and afternoon shift).

Note that as different start and end depot can be specified for a same vehicle, this approach would not necessarily imply a return to the depot for a pause. Instead, a dummy depot with a transition time of zero to each other location could be used to model such case.

Same Vehicle Forward/Backward The forward and the backward trips can be constrained in order to be handled by the same vehicle (Equation (4.17)). Parameter q_r is a boolean value indicating if the forward and the backward trip of request r must be handled by the same vehicle.

$$\forall r \in R \mid q_r = 1 : \; v(A_r^F) = v(A_r^B) \tag{4.17}$$

Alternative Objectives Other objective functions can be considered. For instance, we could be interested in minimizing the accumulated travel time for all the patients (Equation (4.18)). The travel time of a request corresponds to the duration of its activities.

$$\min \left(\sum_{r \in R} d(A_r) \right) \tag{4.18}$$

It is also possible to minimize the maximum travel time (Equation (4.19)). To do so, the maximal duration of the whole set of activities has to be minimized.

$$\min \left(\max_{r \in R} d(A_r) \right) \tag{4.19}$$

Other objective functions are also proposed in [CL03b]. They can be used together inside the same model using either a lexicographic ordering or a Pareto multi objective criterion [NZE05].

4.4 Successor Model

As an alternative to our approach, a successor model was considered. Such models are often employed for VRP problems and similar models were used for solving DARPs using CP [BCL12; JV11]. It uses the circuit constraint mentioned in Section 2.2.1.2 to model the routes of the vehicles as a circuit.

Each trip is represented by two stops which correspond to the place where the patient is loaded and the place where they are unloaded. Each request has then two or four stops depending on whether it is a single trip or a double trip. The successor and the predecessor of each stop are both modeled by a variable indicating the next and the previous stop in the route. Additional stops represent the starting and ending depots of the vehicles. All the routes form a single circuit by assigning the successor of the end depots to the starting depot of the next route. As in [JV11], maximum wait time and vehicle capacity constraints are modeled via auxiliary variables representing the load, serving vehicle, and serving time for each requests. A `circuit` constraint [Lau78] ensures that the successor and predecessor variables form a circuit without sub-tours for each vehicle. The requests that are not serviced are assigned to a same dummy vehicle $\perp$ with infinite capacity and an availability window of $[0; +\infty]$.

4.4.1 Decision Variables

Selection variables ($S(r)$) and activity variables ($s(A_i)$, $d(A_i)$, $e(A_i)$ and $v(A_i)$) from the scheduling model are kept with the following changes: Activity execution variables ($x(A_i)$) are not needed for this model. The dummy vehicle $\perp$ is added to the domain of each vehicle variable.

Additional variables are used for each stop (location to visit). Note that destination stops (dst_r) for requests that have both a forward and a backward trip are duplicated (noted dst'_r) as they correspond to two different steps in the routes of the vehicles. Additionally, two depot stops ($sd_\perp$ and $ed_\perp$) are created for the dummy vehicle $\perp$. The set of all the stops used in the model is noted X^+ and the set of the vehicles is noted $V^+ = V \cup \perp$. The variables for each stop $x \in X^+$ are:

Successor variables $succ(x)$ indicate the next stop in the route. For end depots they are assigned to the start depot of the next route:

$$\forall v \in V^+ : \; succ(ed_v) = \begin{cases} sd_{v+1} & \text{if } v < |V^+| \\ sd_0 & \text{if } v = |V^+| \end{cases} \tag{4.20}$$

Otherwise, their domain is the set of all stops minus the starting depots.

Predecessor variables $pred(x)$ indicate the previous stop in the route. For start depots they are assigned to the end depot of the previous route:

$$\forall v \in V^+ : \; pred(sd_v) = \begin{cases} ed_{v-1} & \text{if } v > 0 \\ ed_{|V^+|} & \text{if } v = 0 \end{cases} \tag{4.21}$$

Otherwise, their domain is the set of all stops minus the end depots.

Arrival variables $arr(x)$ indicate the arrival time at the stop. For depot stops, their domain corresponds to the availability window of the vehicle:

$$\forall v \in V^+ : \begin{cases} arr(sd_v) = [savail_v; eavail_v] \\ arr(ed_v) = [savail_v; eavail_v] \end{cases} \tag{4.22}$$

For other stops, they are views of the corresponding activities starts and ends:

$$\forall r \in R : \begin{cases} arr(org_r) = s(A_r^F) \\ arr(dst_r) = e(A_r^F) - srv_r \\ arr(dst'_r) = s(A_r^B) \\ arr(ret_r) = e(A_r^B) - srv_r \end{cases} \tag{4.23}$$

Departure variables $dep(x)$ are views of the arrival variables that indicate the departure time at the stop.

$$\forall v \in V^+ : \begin{cases} dep(sd_v) = arr(sd_v) \\ dep(ed_v) = arr(ed_v) \end{cases} \tag{4.24}$$

$$\forall r \in R, \forall x \in r : \; dep(x) = arr(x) + srv_r \tag{4.25}$$

Vehicle variables $vcl(x)$ indicate the vehicle that serves the stop. For depots, they are assigned to a single vehicle:

$$\forall v \in V^+ : \begin{cases} vcl(sd_v) = v \\ vcl(ed_v) = v \end{cases} \tag{4.26}$$

For other stops, they are views of the corresponding activities vehicles:

$$\forall r \in R : \begin{cases} vcl(org_r) = v(A_r^F) \\ vcl(dst_r) = v(A_r^F) \\ vcl(dst'_r) = v(A_r^B) \\ vcl(ret_r) = v(A_r^B) \end{cases} \tag{4.27}$$

4.4.2 Constraints and Objective

The *Cumulative Resource* constraint (Equation (4.11)) is kept as in the scheduling model. The *Binding Requests to Activities* constraint (Equation (4.7)) is changed in:

$$\forall r \in R : \; \big(S(r) = 1\big) \equiv \big(v(A_r^F) \neq \bot \wedge v(A_r^B) \neq \bot\big) \tag{4.28}$$

The *Forward and Backward Selection* constraint (Equation (4.8)) is changed in:

$$\forall r \in R : \ v(A_r^F) \neq \perp \equiv v(A_r^B) \neq \perp \tag{4.29}$$

The other constraints from the scheduling model are not used. The objective is kept the same as in the scheduling model (Equation (4.14)). The following constraints are added:

Circuit Two constraints are used to model the routes as a circuit (Equations (4.30) and (4.31)). The *Inverse* constraint (see Section 2.2.1.2) links together the successor and predecessor variables so that they reflect the same links between predecessor and successor in both ways. The *Circuit* constraint enforces that the predecessor and successors form a hamitonian tour (see Section 2.2.1.2).

$$Inverse(pred, succ) \tag{4.30}$$

$$Circuit(succ) \tag{4.31}$$

Time Travel Consistency In order to enforce time travel consistency between predecessors and successors, the following constraints are added:

$$\forall x \in X^+ : \ vcl(x) \neq \perp \implies \begin{cases} arr(x) \geq dep(pred(x)) + trans_{pred(x),x} \\ arr(succ(x)) \geq dep(x) + trans_{x,succ(x)} \end{cases} \tag{4.32}$$

Vehicle Consistency The vehicle consistency between successive stops is enforced by the following constraints:

$$\forall x \in X^+ : \begin{cases} vcl(pred(x)) = vcl(x) \text{if } x \text{ is not a starting depot} \\ vcl(succ(x)) = vcl(x) \text{if } x \text{ is not an end depot} \end{cases} \tag{4.33}$$

Additionally, the vehicle consistency for a same travel is enforced by the constraint:

$$\forall (p, d) \in T : \ vcl(p) = vcl(d) \tag{4.34}$$

4.5 Optional Decision Search

The search tree for both models is explored using a standard branch and bound depth first search. The decision variables are divided into two categories: the *request variables* (Equation (4.7)) and the *activity variables* (Equation (4.6)). For the successor model, stop variables (Equations (4.20) to (4.27)) are also considered as activity variables. Given the main objective of the problem (maximizing the number of served patients), our primal heuristic is to select

patients on the left branches ($S_r = 1$) and discard them on the right branches ($S_r = 0$). Whenever a patient has been selected in a search node, all its related activity variables are subsequently assigned (start time, duration and end time and vehicle) before considering again the next patient selection variable. On the contrary, whenever a patient is not selected ($S_r = 0$ on the right branch), there is no need to consider the other decision variables related to this patient. The idea is to branch on the activity variables only if the related request variable has been selected ($S_r = 1$). Otherwise, no search is performed on the activity variables.

We denote this search strategy as the *Optional Decision Search*. The main asset of this search is that activity variables are branched on only when they are relevant to a solution. It drastically reduces the size of the search tree. An example of search tree is illustrated in Figure 4.5. Algorithm 3 presents the pseudo-code for the method responsible to select the next branching decision.

Algorithm 3: `OptionalDecisionBranching()`

```
Data: decisionVars: Boolean decision variables;
optionalVars: Optional variables;
varsToBranchOn: Optional variables to branch on, initially empty
1 if varsToBranchOn.isEmpty() then
2     currentDecision ← decisionHeuristic(decisionVars); // Selects
        the next decision var to branch on
3     varsToBranchOn ← getLinkedVars(optionalVars, currentDecision);
        // Gets the optional vars linked to the selected decision
4     branch on currentDecision;
5 else
6     currentDecision ← optionalHeuristic(varsToBranchOn); // Selects
        the next optional var to branch on
7     varsToBranchOn.remove(currentDecision);
8     branch on currentDecision;
```

This meta-search strategy for optional activities can be combined with any existing variable-value heuristic. It could also be used for similar applications such as packing problems [DD92].

Example 4.5.1. For example the Rectangle Packing Problem consists in packing as much small rectangles as possible in a large polygon without overlap. This problem can be modeled as such: each small rectangle r is associated to a boolean decision variable $S(r)$ that indicates if it is part of the solution and a series of optional variables that indicate how it arranged in the solution. The optional variables are $x(r)$ and $y(r)$ that indicate the coordinates of one of the corners of the rectangle and $rot(r)$ which indicates its rotation. The optional decision search can be used as such: the main search tree branches on the decision variables; when a decision variable is set to true, a sub-search occurs

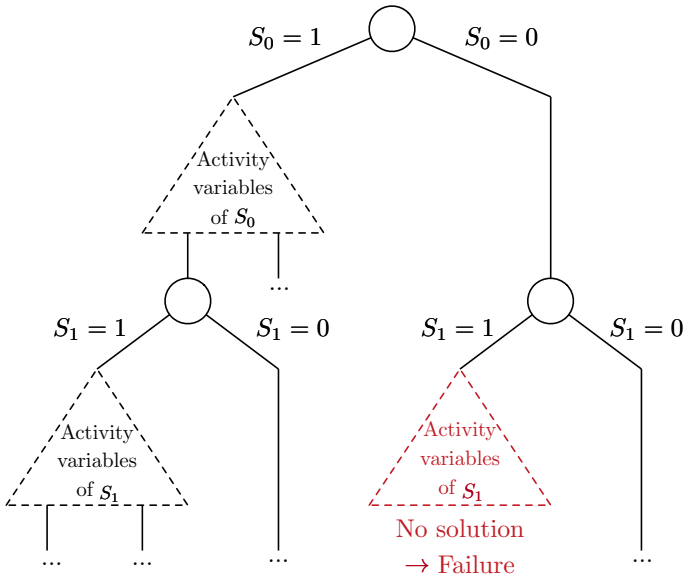

Figure 4.5: Canonical shape of the search tree for two request selection variables (S_0 and S_1)

on its optional variables in order to find a suitable position for the rectangle. An illustration of this problem is given in Figure 4.6.

As a variable heuristic on the request variables we use a Conflict Ordering Search heuristic (COS) [Gay+15]. A conflict is recorded on a request only when it is impossible to assign in the sub-tree all its other activity variables. The fallback heuristic combined with COS is to select the next requests with the highest *minimum slack*, defined as the sum of the minimum duration multiplied by the patient load for its forward and backward activities. The subsearch on the other activity variables follows a *min-domain first fail* strategy for the variable selection and a custom greedy value heuristic based on the type of the corresponding variable which can be a time-related decision or a vehicle choice. In the former case, the heuristic selects the closest time to the corresponding appointment. In the latter case, the vehicle that has the most remaining places is selected.

Large Neighborhood Search In order to boost the performances on large instances, a Large Neighborhood Search (LNS) [Sha98] is also used. At each iteration, a set of request variables is chosen randomly and then relaxed. The other variables are fixed to their value in the last solution. For the request variables that are selected ($S_r = 1$), the corresponding activity variables are also fixed based on the current solution. The remaining unbound variables

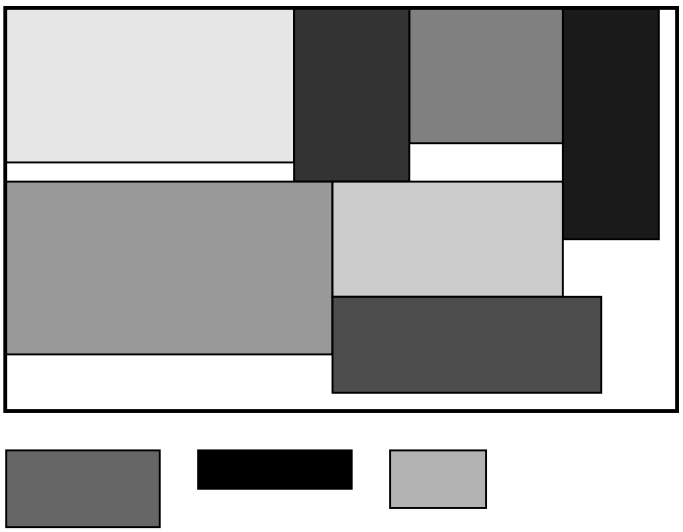

Figure 4.6: Illustration of a rectangle packing problem

form a smaller search space which is explored using the search defined earlier. A new iteration is started when the reduced search space is completely explored or once a fixed number of backtracks is reached. The usage of an ALNS approach was considered but discarded as preliminary experiments showed no improvement compared to the basic LNS with a conflict ordering search and the optional decision search.

4.6 Experimental Results

This section evaluates the performance of the models on synthetic and real instances. The models tested are referred as the *Scheduling with Optional Decision Search* (SCHED+ODS) and the *Successor* (SUCC) approaches. They correspond to the models described in Sections 4.3 and 4.4 The objective considered is to maximize the number of requests satisfied (Equation (4.14)).

4.6.1 Approaches Considered

Our two models are compared with three other approaches: a greedy search, the same CP scheduling model without the Optional Decision search and a similar scheduling model implemented in CP Optimizer. Note that the *Successor* approach was only considered with the optional decision search as the version with a classic search did not perform well enough during preliminary experiments.

Greedy Search (GREEDY) It mimics the manual decision process used by the non-profit organization. It consists in selecting first the requests having the earliest starting time and choosing for them the closest compatible vehicle. The idea is to minimize the time between the trips of each vehicle across the requests. Each trip is inserted at the earliest possible time such that later trips can be inserted with more flexibility. If a trip cannot be inserted, the request is discarded.

This approach is sub-optimal as it considers only one trip at a time and thus the vehicles will only take one patient in charge at the same time. However, it is fast and close to the process that would be used by a human operator in order to find a solution manually. Note that this greedy search does not take into account the waiting time of the patients or other objectives. It is thus only comparable with models that maximize the number of requests taken.

CP Optimizer implementation (CPO) The scheduling model has been implemented in IBM CP Optimizer in order to compare our search with the default search proposed by this solver. This search combines an adaptive LNS with a failure directed search (FDS) strategy [VLS15]. In order to accommodate the solver, the capacity constraints of vehicles are modeled using *cumul* functions from CP optimizer.

Scheduling Model with Simple Search (SCHED) It corresponds to the model presented in Section 4.3 without the *Optional Decision search* heuristic. Additional reified constraints assign the activity variables to a default value when a request is not served in order to avoid wasting time searching on activity variables when the corresponding request is not selected.

4.6.2 Datasets Used

The experiments are based on two datasets, a synthetic and a real one. The synthetic dataset has been randomly generated based on the characteristics of the problem. Synthetic instances are classified according to their size, expressed by the number of requests ($|R|$), available vehicles ($|V|$) and health centres ($|H|$) as well as their difficulty which is related to the number of constraints and the availability of vehicles. Note that the while increasing R and H has a direct impact on the size and thus difficulty of the instance to solve, increasing V actually makes the instance easier as more vehicles are available to transport the patients.

The real dataset has been provided by the non-profit organization. It corresponds to one month of exploitation with one instance per day. Each of them contains the requests received for the day, the vehicles available and

the distance matrix that was computed using the direct distance between geographical coordinates. In terms of difficulty and number of constraints, this dataset is situated between the easy and medium synthetic datasets.

The instances are named following this convention: PTP−[dataset type]_-[number of hospitals]_[number of vehicles]_[number of requests]

4.6.3 Experimental Protocol

Experiments have been carried out on an AMD Opteron 6176 processor (2300 MHz). Execution time for a run is limited to 1800 seconds and memory consumption to 6 GB. The greedy search has been implemented in Scala and the OscaR solver [Osc12] is used for the other models except for the CPO model that has been modeled and solved with the java API of the academic version of IBM ILOG CPLEX CP Optimizer V12.8. For the reproducibility of results, the models, the synthetic dataset and the random generator are available online on CSPLib [Tho+].

The backtrack limit and relaxation size of the LNS are adaptive parameters, initially fixed to respectively 1000 failures and 10 requests. The backtrack limit is increased by 20% when 100 consecutive iterations have failed to find an new solution and to completely explore the search. The relaxation size is increased by 20% when the relaxed search space is completely explored for 50 consecutive iterations. Search parameters are set to their defaults for CPO. The greedy solution is considered as the first solution of the LNS for each method.

Given the random nature of approaches based on LNS (SUCC, CPO, SCHED and SCHED+ODS), 5 runs for each instance with a different seed have been performed and the best solution obtained is recorded. The greedy search (GREEDY) is ran only once due to its deterministic nature. The models are also compared using the improvement ratio (ρ_m) of a method (m) defined as the relative improvement of the solution obtained with the method (x_m) compared to the solution found using the greedy search (x_{GREEDY}).

$$\rho_m = \frac{x_m - x_{\text{GREEDY}}}{x_{\text{GREEDY}}} \tag{4.35}$$

4.6.4 Initial Results

Results for synthetic instances are reported in Table 4.2. Results on real world data instances are reported in Table 4.3. Instances are ordered by their difficulty and the number of patients ($|R|$). The best solution obtained for each instance is also reported. The number of patients serviced is considered as the objective value. As the relaxation size is adaptive, it can eventually grow to 100%. In this case, if the search space is completely explored, the solution

is proven optimal. Besides, if all the patients are serviced, the upper bound is reached and the solution is also proven optimal. The dominating model(s) is(are) highlighted in green for each instance. If a model has not been able to improve the initial solution found by the greedy approach, it is highlighted in red.

Table 4.2: Experimental results for the synthetic instances (ρ is the improvement ratio in percent, * indicates that the solution has been proven optimal)

Instances						GREEDY	SUCC		CPO		SCHED		SCHED+ODS	
Difficulty	Name	$\|H\|$	$\|V\|$	$\|R\|$	BestSol	Sol	Sol	ρ	Sol	ρ	Sol	ρ	Sol	ρ
Easy	RAND-E-1	4	2	16	*15	14	15	7.1	*15	7.1	*15	7.1	15	7.1
	RAND-E-2	8	4	32	*32	32	32	0.0	*32	0.0	*32	0.0	*32	0.0
	RAND-E-3	12	5	48	*28	26	26	0.0	*28	7.7	28	7.7	*28	7.7
	RAND-E-4	16	6	64	62	58	61	5.2	59	1.7	62	6.9	62	6.9
	RAND-E-5	20	8	80	74	72	73	1.4	72	0.0	73	1.4	74	2.8
	RAND-E-6	24	9	96	95	91	93	2.2	92	1.1	92	1.1	95	4.4
	RAND-E-7	28	10	112	106	100	101	1.0	100	0.0	103	3.0	106	6.0
	RAND-E-8	32	12	128	*128	127	*128	0.8	127	0.0	*128	0.8	*128	0.8
	RAND-E-9	36	14	144	142	141	142	0.7	141	0.0	142	0.7	142	0.7
	RAND-E-10	40	16	160	157	154	154	0.0	157	1.9	157	1.9	157	1.9
Medium	RAND-M-1	8	2	16	*12	8	9	12.5	11	37.5	*12	50.0	11	37.5
	RAND-M-2	16	3	32	19	16	18	12.5	17	6.3	19	18.8	19	18.8
	RAND-M-3	24	4	48	32	25	25	0.0	26	4.0	30	20.0	32	28.0
	RAND-M-4	32	4	64	37	25	25	0.0	33	32.0	35	40.0	37	48.0
	RAND-M-5	40	5	80	55	45	45	0.0	48	6.7	51	13.3	55	22.2
	RAND-M-6	48	5	96	52	36	40	11.1	40	11.1	50	38.9	52	44.4
	RAND-M-7	56	6	112	63	46	47	2.2	48	4.3	63	37.0	63	37.0
	RAND-M-8	64	8	128	83	65	70	7.7	65	0.0	81	24.6	83	27.7
	RAND-M-9	72	8	144	81	62	62	0.0	64	3.2	72	16.1	81	30.6
	RAND-M-10	80	9	160	99	73	75	2.7	75	2.7	88	20.5	99	35.6
Hard	RAND-H-1	16	2	16	8	7	7	0.0	8	14.3	8	14.3	8	14.3
	RAND-H-2	32	3	32	19	15	15	0.0	18	20.0	19	26.7	17	13.3
	RAND-H-3	48	4	48	32	18	19	5.6	23	27.8	32	77.8	29	61.1
	RAND-H-4	64	4	64	23	10	12	20.0	22	120.0	20	100.0	23	130.0
	RAND-H-5	80	5	80	42	29	31	6.9	29	0.0	38	31.0	42	44.8
	RAND-H-6	96	5	96	38	22	22	0.0	27	22.7	38	72.7	38	72.7
	RAND-H-7	112	6	112	39	25	27	8.0	32	28.0	37	48.0	39	56.0
	RAND-H-8	128	8	128	75	57	63	10.5	61	7.0	71	24.6	75	31.6
	RAND-H-9	144	8	144	72	50	54	8.0	53	6.0	67	34.0	72	44.0
	RAND-H-10	160	8	160	72	46	48	4.3	50	8.7	63	37.0	72	56.5

Let us first focus on synthetic instances. As we can see, the scheduling model with the Optional Decision search (SCHED+ODS) obtains the best solution for almost all the tests, even when the optimum is not reached. The improvement ratio is up to 130% compared to the greedy solution. Interestingly, the performance of the scheduling model is correlated with the difficulty of instances: the improvement gap increases when the instances are getting harder. The greedy search (GREEDY) gives poor solutions when the problem is strongly constrained. Results regarding the scheduling model with the simple search (SCHED) shows the interest of the custom search.

The successor model (SUCC) is outperformed by the scheduling models. This is expected as the successor model has a larger search space due to the additional decisions variables compared to the scheduling model. Furthermore, the successor approach makes the insertion of new stops in routes more dif-

Table 4.3: Experimental results for the real instances (ρ is the improvement ratio in percent, * indicates that the solution has been proven optimal)

Instances					GREEDY	SUCC		CPO		SCHED		SCHED+ODS	
Name	$\|H\|$	$\|V\|$	$\|R\|$	BestSol	Sol	Sol	ρ	Sol	ρ	Sol	ρ	Sol	ρ
REAL-1	1	9	2	*2	2	*2	0.0	*2	0.0	*2	0.0	*2	0.0
REAL-2	1	9	2	*2	2	*2	0.0	*2	0.0	*2	0.0	*2	0.0
REAL-3	3	9	3	*1	1	*1	0.0	*1	0.0	*1	0.0	*1	0.0
REAL-4	2	9	4	*4	4	*4	0.0	*4	0.0	*4	0.0	*4	0.0
REAL-5	5	9	21	*21	21	*21	0.0	*21	0.0	*21	0.0	*21	0.0
REAL-6	5	9	22	*22	22	*22	0.0	*22	0.0	*22	0.0	*22	0.0
REAL-7	5	9	23	*23	23	*23	0.0	*23	0.0	*23	0.0	*23	0.0
REAL-8	7	9	24	*24	24	*24	0.0	*24	0.0	*24	0.0	*24	0.0
REAL-9	15	9	45	*44	44	44	0.0	*44	0.0	*44	0.0	*44	0.0
REAL-10	26	9	99	*98	98	98	0.0	*98	0.0	*98	0.0	*98	0.0
REAL-11	22	9	100	91	87	89	2.3	87	0.0	90	3.4	91	4.6
REAL-12	32	9	101	*100	97	98	1.0	97	0.0	*100	3.1	99	2.1
REAL-13	37	9	110	103	97	98	1.0	97	0.0	100	3.1	103	6.2
REAL-14	28	9	111	*102	99	99	0.0	100	1.0	100	1.0	*102	3.0
REAL-15	35	9	122	110	94	97	3.2	94	0.0	102	8.5	110	17.0
REAL-16	36	9	123	108	107	107	0.0	108	0.9	108	0.9	108	0.9
REAL-17	42	9	128	114	103	103	0.0	105	1.9	105	1.9	114	10.7
REAL-18	31	9	130	121	112	115	2.7	113	0.9	115	2.7	121	8.0
REAL-19	34	9	131	114	103	107	3.9	103	0.0	108	4.9	114	10.7
REAL-20	34	9	134	118	106	107	0.9	106	0.0	108	1.9	118	11.3
REAL-21	39	9	136	119	108	112	3.7	108	0.0	114	5.6	119	10.2
REAL-22	31	9	138	121	113	117	3.5	113	0.0	117	3.5	121	7.1
REAL-23	31	9	139	121	113	113	0.0	113	0.0	115	1.8	121	7.1
REAL-24	37	9	139	110	103	103	0.0	104	1.0	106	2.9	110	6.8
REAL-25	39	9	139	125	118	118	0.0	121	2.5	121	2.5	125	5.9
REAL-26	38	9	140	119	107	107	0.0	109	1.9	115	7.5	119	11.2
REAL-27	35	9	147	129	120	121	0.8	120	0.0	126	5.0	129	7.5
REAL-28	34	9	151	131	115	116	0.9	115	0.0	121	5.2	131	13.9
REAL-29	39	9	155	127	117	119	1.7	117	0.0	123	5.1	127	8.5
REAL-30	41	9	159	131	115	115	0.0	119	3.5	121	5.2	131	13.9

ficult as it requires to change the value of the successor variables forming the routes in addition to the vehicle variable. This limits the effectiveness of the LNS.

Concerning the CP Optimizer model (CPO), it is also outperformed by the two other scheduling approaches. Such results could be explained by the default search used in CPO model: it is generic and not designed for this specific problem. Another point to take into account is that this model uses generic variables and structures while CP Optimizer provides dedicated variables for scheduling problems such as interval variables and sequence variables. Thus, the performances could certainly be improved by using these dedicated representations at the cost of making the model harder to adapt in other solvers or frameworks. It is also important to point out that on harder instances, it tends to perform better than the successor model. This could indicate that the model used contributes more to the effectiveness of the approach than the search method. Note that as the CPO approach is based on another solver,

other factors could also influence the performances.

Similar results are observed for the real instances. The scheduling model with the Optional Decision search is dominating again. However, the improvement ratio is now up to 17% only. It happens because such real instances are easier to solve compared to the medium and difficult synthetic instances. It shows both the pertinence of the scheduling model and the search framework we introduced.

We also considered the waiting time minimization (Eq. 4.18) as a secondary objective using a lexicographical search. However, it yielded only minor improvements regarding the solution obtained using the main objective. It mainly occurs because the value heuristic used already ensures that solutions minimizing the waiting time are tried first.

4.6.5 Comparison with Liu et al.

This section presents the comparison of the approaches previously discussed with the adaptation of the model proposed in [LAB18] to the PTP. The new model, which is described in section 4.2 was run on both datasets of instances using the same conditions as those detailed in Section 4.6.3. It is referred as CPOLIU. The results are displayed in Tables 4.4 and 4.5.

As it can be seen, the CPOLIU model greatly outperforms the other models as it is able to find a better solution on all the instances for which the optimum has not been reached by the other models. The comparison between the CPO and CPOLIU approaches is especially interesting as both use the same default adaptive LNS search of CP Optmizer. This is mainly due to the fact that Liu et al. model combines a scheduling approach with the powerful dedicated variables of CP Optimizer.

4.7 Conclusion

In this chapter, the Patient Transportation Problem (PTP) was presented and formalized. Several CP models designed to tackle this problem were detailed and compared. The contributions in this chapter are:

- The definition and formalization of the PTP which is a real industrial problem (Section 4.1).
- The scheduling based model proposed in Section 4.3.
- The successor model proposed in Section 4.4.
- The adaptation of the STP CP model from [LAB18] to the PTP (Section 4.2).

Table 4.4: Comparison of performances on the synthetic instances (ρ is the improvement ratio in percent, * indicates that the solution has been proven optimal)

Instances						GREEDY	CPO		SCHED+ODS		CPOLIU	
Difficulty	Name	$\|H\|$	$\|V\|$	$\|R\|$	BestSol	Sol	Sol	ρ	Sol	ρ	Sol	ρ
easy	RAND-E-1	4	2	16	*15	14	*15	7.1	*15	7.1	15	7.1
	RAND-E-2	8	4	32	*32	32	*32	0.0	*32	0.0	32	0.0
	RAND-E-3	12	5	48	*28	26	*28	7.7	*28	7.7	28	7.7
	RAND-E-4	16	6	64	64	58	59	1.7	62	6.9	64	6.7
	RAND-E-5	20	8	80	79	72	72	0.0	74	2.8	79	8.2
	RAND-E-6	24	9	96	96	91	92	1.1	95	4.4	96	5.5
	RAND-E-7	28	10	112	112	100	100	0.0	106	6.0	112	9.8
	RAND-E-8	32	12	128	*128	127	127	0.0	128	0.8	128	0.8
	RAND-E-9	36	14	144	144	141	141	0.0	142	0.7	144	1.4
	RAND-E-10	40	16	160	160	154	157	1.9	157	1.9	160	1.9
medium	RAND-M-1	8	2	16	*12	8	11	37.5	11	37.5	12	50.0
	RAND-M-2	16	3	32	25	16	17	6.3	19	18.8	25	38.9
	RAND-M-3	24	4	48	43	25	26	4.0	32	28.0	43	65.4
	RAND-M-4	32	4	64	50	25	33	32.0	37	48.0	50	100.0
	RAND-M-5	40	5	80	68	45	48	6.7	55	22.2	68	51.1
	RAND-M-6	48	5	96	60	36	40	11.1	52	44.4	60	50.0
	RAND-M-7	56	6	112	77	46	48	4.3	63	37.0	77	67.4
	RAND-M-8	64	8	128	109	65	65	0.0	83	27.7	109	51.4
	RAND-M-9	72	8	144	109	62	64	3.2	81	30.6	109	67.7
	RAND-M-10	80	9	160	129	73	75	2.7	99	35.6	129	72.0
hard	RAND-H-1	16	2	16	13	7	8	14.3	8	14.3	13	85.7
	RAND-H-2	32	3	32	23	15	18	20.0	17	13.3	23	53.3
	RAND-H-3	48	4	48	43	18	23	27.8	29	61.1	43	126.3
	RAND-H-4	64	4	64	36	10	22	120.0	23	130.0	36	227.3
	RAND-H-5	80	5	80	64	29	29	0.0	42	44.8	64	113.3
	RAND-H-6	96	5	96	58	22	27	22.7	38	72.7	58	163.6
	RAND-H-7	112	6	112	59	25	32	28.0	39	56.0	59	136.0
	RAND-H-8	128	8	128	108	57	61	7.0	75	31.6	108	86.2
	RAND-H-9	144	8	144	101	50	53	6.0	72	44.0	101	87.0
	RAND-H-10	160	8	160	99	46	50	8.7	72	56.5	99	106.3

- The Optional Decision Search (ODS) heuristic that is designed to provide an efficient search for problems presenting optional decisions such as the PTP or the rectangle packing problem (Section 4.5).
- An experimental comparison of the different approaches studied on both synthetic and real instances of the PTP (Section 4.6).

The experimental results highlight the importance of the modeling for the resolution of such problems. The good results of the Liu et al. approach that uses the components of CP optimizer dedicated to scheduling problems demonstrate that the usage of dedicated variables and constraints can greatly improve the resolution of complicated problems such as the PTP. However, such specific variables can be hard to implement compared to generic ones which makes them less frequent in most CP solvers. Nevertheless, sequence variables can be used to model a large range of problems and may provide benefits in terms of modeling and performances. This led us to study their usage and propose new implementations for them in the next chapter.

Table 4.5: Comparison of performances on the real instances (ρ is the improvement ratio in percent, * indicates that the solution has been proven optimal)

Instances					GREEDY	CPO		SCHED+ODS		CPOLIU	
Name	$\|H\|$	$\|V\|$	$\|R\|$	BestSol	Sol	Sol	ρ	Sol	ρ	Sol	ρ
REAL-1	1	9	2	*2	2	*2	0.0	*2	0.0	2	0.0
REAL-2	1	9	2	*2	2	*2	0.0	*2	0.0	2	0.0
REAL-3	3	9	3	*1	1	*1	0.0	*1	0.0	1	0.0
REAL-4	2	9	4	*4	4	*4	0.0	*4	0.0	4	0.0
REAL-5	5	9	21	*21	21	*21	0.0	*21	0.0	21	0.0
REAL-6	5	9	22	*22	22	*22	0.0	*22	0.0	22	0.0
REAL-7	5	9	23	*23	23	*23	0.0	*23	0.0	23	0.0
REAL-8	7	9	24	*24	24	*24	0.0	*24	0.0	24	0.0
REAL-9	15	9	45	*44	44	*44	0.0	*44	0.0	44	0.0
REAL-10	26	9	99	*98	98	*98	0.0	*98	0.0	98	0.0
REAL-11	22	9	100	91	87	87	0.0	91	4.6	92	3.4
REAL-12	32	9	101	*100	97	97	0.0	99	2.1	100	2.0
REAL-13	37	9	110	103	97	97	0.0	103	6.2	107	9.2
REAL-14	28	9	111	*102	99	100	1.0	*102	3.0	*102	3.0
REAL-15	35	9	122	110	94	94	0.0	110	17.0	118	21.6
REAL-16	36	9	123	108	107	108	0.9	108	0.9	117	9.3
REAL-17	42	9	128	114	103	105	1.9	114	10.7	122	18.4
REAL-18	31	9	130	121	112	113	0.9	121	8.0	126	12.5
REAL-19	34	9	131	114	103	103	0.0	114	10.7	121	13.1
REAL-20	34	9	134	118	106	106	0.0	118	11.3	127	18.7
REAL-21	39	9	136	119	108	108	0.0	119	10.2	126	12.5
REAL-22	31	9	138	121	113	113	0.0	121	7.1	131	13.9
REAL-23	31	9	139	121	113	113	0.0	121	7.1	133	17.7
REAL-24	37	9	139	110	103	104	1.0	110	6.8	120	16.5
REAL-25	39	9	139	125	118	121	2.5	125	5.9	134	12.6
REAL-26	38	9	140	119	107	109	1.9	119	11.2	132	20.0
REAL-27	35	9	147	129	120	120	0.0	129	7.5	137	13.2
REAL-28	34	9	151	131	115	115	0.0	131	13.9	145	23.9
REAL-29	39	9	155	127	117	117	0.0	127	8.5	146	21.7
REAL-30	41	9	159	131	115	119	3.5	131	13.9	146	25.9

Sequence Variables

5

Following the results of the different approaches considered to solve the Patient Transportation Problem, we realized the need for a better modeling of the problem. In particular, the inter-activity time travel consistency constraints defined in Equation 4.9 have an important impact on performances. This led us to consider the usage of sequence variables to model the tour of the vehicles in the PTP and in other DARP variants. This research direction was also considered promising in the context of the PRESupply project as sequence variables are a useful modeling component that can be used for a large range of optimisation problems.

This chapter describes two possible implementations of a sequence variable for modeling and solving the PTP: the *Prefix Sequence Variable* (PSV) and the *Insertion Sequence Variable* (ISV). Both domain representation include the subset bound domain [Ger97] for set variables. This allows to represent optional elements in the domain and prevents a repetition of the same element at different positions in the sequence. The set domain is extended with an internal sequence that can be grown during the search. In addition, the insertion sequence variable considers a set of possible insertions that can be used to grow the internal sequence. By letting the constraints remove impossible elements or insertions, the search space is pruned by restricting the set of possible sequences. We describe several important global constraints on the sequence variables for modeling the DARP and PTP:

1. The `First` and `Last` constraints ensure that an element is first or last in a sequence.

2. The `Dependency` constraint enforces a dependency between several elements of a sequence: they must either be all part of the sequence or all excluded.

3. The `Precedence` constraint ensures that elements of a sequence follow a given order.

4. The `Sequence Allocation` constraint links elements that are possible in several sequence variables with integer variables indicating the sequence in which the element is part of.

5. The `Transition Times` constraint links a sequence variable with time interval variables to take into account a transition time matrix between consecutive elements in the sequence.

6. The `Cumulative` constraint ensures that the load profile does not exceed a fixed capacity when pairs of elements in the sequence represent the load and discharge on a vehicle.

7. the `Max Distance` constraint links an integer variable with the total distance between elements a sequence for which a transition matrix is given.

5.1 Related Work

In [JV11], the authors propose a constraint-based approach called *LNS-FFPA* to solve DARPs with a cost objective and show that it outperforms other state-of-the-art approaches. While highly efficient, the LNS-FFPA algorithm is difficult to adapt to other variants of the DARP such as the PTP. Indeed, the approach is not declarative since some constraints are enforced with the search. Furthermore, the model is not able to deal with optional visits that occur in the PTP and similar problems.

As demonstrated in Section 4.6.5, the approach of [LAB18], which uses IBM ILOG CP Optimizer solver [Lab+18] is the most promising to solve the PTP. It makes use of the sequence variables from CP Optimizer to decide the order of visits in each vehicle.

The high level functionalities and constraints related to sequence variables of CP Optimizer have been briefly described in [LR08; Lab+09]. Unfortunately, no details are given on the implementation of such variables and the filtering algorithms of the constraints in the literature. According to the API and documentation available at [IBM19a; IBM19b], the sequence variable of CP Optimizer is based on a *Head-Tail Sequence Graph* structure. It consists of maintaining separate growing head and tail sub-sequences. Interval variables not yet sequenced can be added either at the end of the head or at the beginning of the tail. When no more interval variable can be added, both sub-sequences are joined to form the final sequence. Note that as interval variables are optional in CP Optimizer, it is possible to add undecided interval variables to the head or the tail. If these variables are later become absent, they are removed from the head or the tail. The Head-Tail Sequence Graph structure is illustrated in Figure 5.1. Google OR-Tools [PF19a] also propose sequence variables [PF19b] with the same approach as CP Optimizer.

The sequence variables proposed in this thesis differ from the one of CP Optimizer in the following ways:

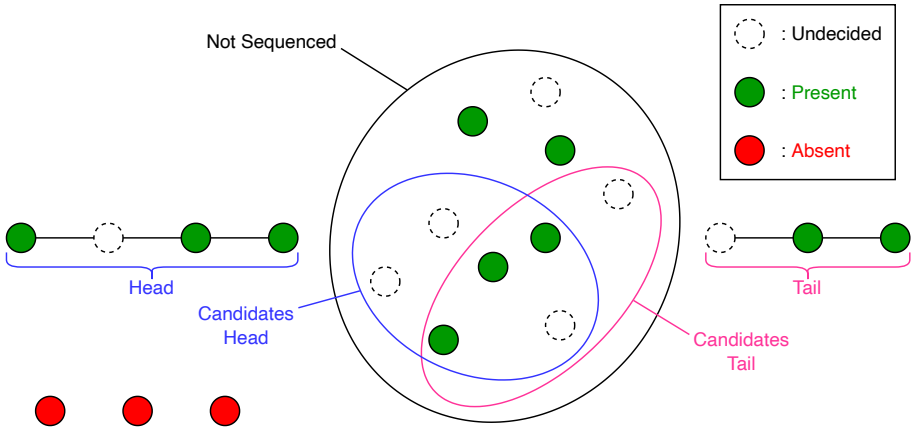

Figure 5.1: Structure of the Head-Tail Sequence Graph

1. both implementations are generic and usable in a large variety of problems. In particular, the variables are independent of the notion of time intervals;

2. for the insertion variable, insertions are allowed at any point in the sequence which allows flexible modeling and search;

3. Both implementations make use of a set domain to keep track of the elements that are either required, excluded or optional in the sequence; in addition, the insertion sequence variable keeps track of the possible insertions for each element inside its domain which allows advanced propagation techniques.

In [Har+15a], the authors discuss the usage of a path variable in the context of Segment Routing Problems. Their implementation is based on a growing prefix to which candidates elements can be appended.

5.2 Preamble

The most basic definition of a *sequence* is a collection of elements in which order matters. Sequences are used in many fields in mathematics and computer science and their precise definition may vary following the domains. What is referred as a sequence in this thesis is more restrained than the general definition in two ways: First, the sequence is finite. Second, each element is unique and no repetition of the same element is allowed in the sequence.

Thus, the notion of sequence considered in this thesis can be seen as an ordered set. Additionally let us note the difference between a *sequence* (noted $\overrightarrow{S}$) which refers to a fixed mathematical object (Definition 9) and a *sequence variable* (noted Sq) which is a CP variable which domain is a set of sequences (Definition 10).

Definition 9. In formal terms, we denote by $\overrightarrow{S}$ a *sequence* without duplicates over X ($S \subseteq X$). The sequence $\overrightarrow{S}$ defines an order over the elements of S. Each element of X is unique and can appear only once in S.

Note that not necessarily all elements of X must appear in $\overrightarrow{S}$. The set of all sequences of X is denoted by $\overrightarrow{\mathcal{P}}(X)$. Let a and b be two elements of S. The relation *a precedes b in* $\overrightarrow{S}$ is noted $a \overset{\overrightarrow{S}}{\prec} b$ or $a \prec b$ when it is clear from the context that the relation applies in regards to $\overrightarrow{S}$. The relation *a directly precedes b in* $\overrightarrow{S}$ is noted $a \overset{\overrightarrow{S}}{\longrightarrow} b$ or $a \rightarrow b$ when clear from the context. In this case, b is called the *successor* of a and a is called the *predecessor* of b in $\overrightarrow{S}$.

Example 5.2.1. Let us consider the sequence $\overrightarrow{S} = (1, 2, 3)$. The following relations between couples of elements hold: $1 \prec 2, 1 \prec 3, 2 \prec 3, 1 \rightarrow 2$ and $2 \rightarrow 3$.

Definition 10. A sequence variable Sq on a set of elements X is a variable which domain $D(Sq)$ is composed of sequences $\overrightarrow{S} \in \overrightarrow{\mathcal{P}}(X)$.

Definition 11. The relation *"$\overrightarrow{A}$ is the prefix of $\overrightarrow{B}$"* between two sequences $\overrightarrow{A}$, $\overrightarrow{B} \in \overrightarrow{\mathcal{P}}(X)$ is denoted by the operator $\sqsubseteq$ and defined as

$$\overrightarrow{A} \sqsubseteq \overrightarrow{B} \iff A \subseteq B \wedge \overrightarrow{B} = \overrightarrow{A} + \overrightarrow{B \setminus A} \tag{5.1}$$

where the operator "+" denotes the concatenation of two sequences.

For example, $\overrightarrow{A} = (1, 3, 7)$ is the prefix of $\overrightarrow{B} = (1, 3, 7, 9, 4)$: $\overrightarrow{A} \sqsubseteq \overrightarrow{B}$. The prefix relation is a partial order over $\overrightarrow{\mathcal{P}}(X)$ and the structure $(\overrightarrow{\mathcal{P}}(X), \sqsubseteq)$ is not a lattice.

Definition 12. A sequence $\overrightarrow{S}'$ is a *super-sequence* of $\overrightarrow{S}$ if

$$S \subseteq S' \wedge \forall a, b \in S, a \overset{\overrightarrow{S}}{\prec} b \implies a \overset{\overrightarrow{S'}}{\prec} b \tag{5.2}$$

This relationship is noted $\overrightarrow{S} \subseteq \overrightarrow{S}'$. Conversely, $\overrightarrow{S}$ is a *sub-sequence* of $\overrightarrow{S}'$. In other words, a sequence is a super-sequence of another if it contains all the elements of the sub-sequence and they appear in the same order. Note that other elements may appear in the super-sequence, including between elements of the sub-sequence.

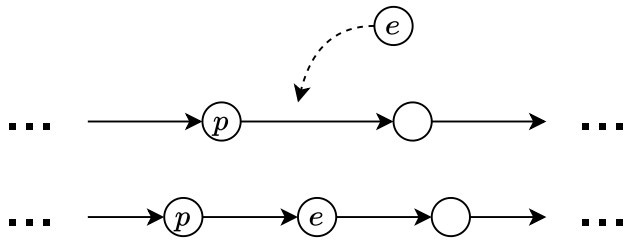

Figure 5.2: Insertion of an element e after p

Definition 13. The *insertion* operation $insert(\overrightarrow{S}, e, p)$ consists in inserting the element e in the sequence $\overrightarrow{S}$ directly after the element p where $e \in X \setminus S$ and $p \in S$. Performing this operation results in a super-sequence $\overrightarrow{S}'$ of $\overrightarrow{S}$ such that $S' = S \cup \{e\}$ and $p \xrightarrow{\overrightarrow{S}'} e$.

The operation is also noted $\overrightarrow{S} \underset{(e,p)}{\Longrightarrow} \overrightarrow{S}'$. It is illustrated in Figure 5.2.

Example 5.2.2. The operation $insert(\overrightarrow{S}, 4, 2)$ applied on the sequence $\overrightarrow{S} = (1, 2, 3)$ results in a sequence $\overrightarrow{S}' = (1, 2, 4, 3)$.

The insertion of an element e at the beginning of a sequence or in an empty sequence is defined as $insert(\overrightarrow{S}, e, \alpha)$. The symbol α indicates the beginning of the sequence. An insertion in a sequence $\overrightarrow{S}$ is thus characterized by a tuple (e, p) where $e \notin S$ and $p \in S \vee p = \alpha$. Note that the insertion of an element e after a predecessor $p \in S$ is equivalent to posting the following constraints:

$$\forall \overrightarrow{S}' \in D(Sq), a \overset{\overrightarrow{S}'}{\prec} e \overset{\overrightarrow{S}'}{\prec} s, \text{ where } s \in S \mid a \xrightarrow{\overrightarrow{S}} s \tag{5.3}$$

Given I, a set of tuples, each corresponding to a potential insertion in $\overrightarrow{S}$,

Definition 14. The *one-step derivation* $\overrightarrow{S} \underset{I}{\Longrightarrow} \overrightarrow{S}'$ between a sequence $\overrightarrow{S}$ and a super-sequence S' is defined as

$$\overrightarrow{S} \underset{I}{\Longrightarrow} \overrightarrow{S}' \iff \exists i = (e, p) \in I \mid \overrightarrow{S} \underset{i=(e,p)}{\Longrightarrow} \overrightarrow{S}' \tag{5.4}$$

In other words, the sequence S is transformed into S' by applying one possible insertion from I.

Definition 15. More generally, *zero or more steps derivation* is defined as

$$\overrightarrow{S} \overset{*}{\underset{I}{\Longrightarrow}} \overrightarrow{S}' \equiv \overrightarrow{S} = \overrightarrow{S}' \vee \left(\exists i \in I \mid \overrightarrow{S} \underset{i}{\Longrightarrow} \overrightarrow{S}'' \wedge \overrightarrow{S}'' \overset{*}{\underset{I \setminus \{i\}}{\Longrightarrow}} \overrightarrow{S}' \right) \tag{5.5}$$

Note that I may contain tuples that do not correspond to a possible insertion in $\overrightarrow{S}$ but instead to a possible insertion in a super-sequence $\overrightarrow{S}'$ of $\overrightarrow{S}$. Also note that several successions of insertions in I may lead to a same super-sequence.

Example 5.2.3. Let us consider the sequence $\overrightarrow{S} = (1, 2, 3)$ and the set of insertions $I = \{(4, 1), (4, 2), (5, 2), (5, 4)\}$. We have that $\overrightarrow{S} \overset{*}{\underset{I}{\Longrightarrow}} \overrightarrow{S}' = (1, 2, 4, 5, 3)$ since it can be obtained with consecutive derivations over I : $(1, 2, 3) \underset{(4,2)}{\Longrightarrow} (1, 2, 4, 3) \underset{(5,4)}{\Longrightarrow} (1, 2, 4, 5, 3)$. Note that $\overrightarrow{S}'$ can also be obtained by inserting first $(5, 2)$ and $(4, 2)$ after. If $(5, 2)$ was not part of I, the relation would still hold as there would still be a possibility to obtain $\overrightarrow{S}'$ with only $(4, 2)$ and $(5, 4)$. Finally, it is possible to have other insertions in I that are not part of any derivation to $\overrightarrow{S}'$ such as $(4, 1)$.

Two different implementations of a sequence variable are considered in this thesis. The *prefix* sequence variable associates a set domain with a growing list of ordered elements. The set tracks which elements are possible, required or excluded in the sequence. The list tracks the elements that are already sequenced. Elements that are added to the sequence can only added at the end of the growing list. The *insertion* sequence variable extends the idea of the growing sequence variable by allowing insertions of new elements at any point among the already sequenced elements. To do so, in addition of the set domain and the growing list, possible insertions are tracked for each non-sequenced element. The prefix sequence variable is defined in Section 5.3 and the insertion sequence variable is detailed in Section 5.4.

5.3 Prefix Sequence Variable

Definition 16. A prefix sequence variable Sq on a set X is a variable whose domain is represented by a tuple $\langle \overrightarrow{S}, P, R, E \rangle$ where $\langle P, R, E \rangle$ is the domain of a set variable on X (see Section 2.2.1.1) and $\overrightarrow{S} \in \overrightarrow{\mathcal{P}}(R)$. Sq is bound if P is empty and $|S| = |R|$. The domain of Sq, also noted $D(Sq)$, is defined as

$$\langle \overrightarrow{S}, P, R, E \rangle \equiv \left\{ \overrightarrow{S}' \in \overrightarrow{\mathcal{P}}(P \cup R) \mid R \subseteq S' \wedge \overrightarrow{S} \sqsubseteq \overrightarrow{S}' \right\}$$

$\overrightarrow{S}$ corresponds to the growing internal sequence, P to the set of *Possible* elements (that have not been decided yet), R to the set of *Required* elements (that must be part of Sq) and E to the set of *Excluded* elements (That cannot be part of Sq). Note that P, R and E form a partition of X. The variable is bound if all elements in P have been moved to either R or E and all elements

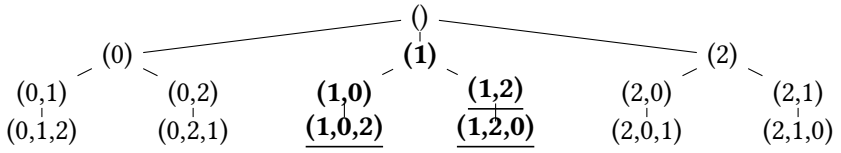

Figure 5.3: Initial domain $\langle (), \{0,1,2\}, \emptyset, \emptyset \rangle$ of Sq over $X = \{0,1,2\}$ and intermediate domain $\langle (1), \{0,2\}, \{1\}, \emptyset \rangle$ (in bold).

of R are part of the internal sequence $\overrightarrow{S}$. Initially, all elements of the domain are optional ($\in P$). During the search, elements can be set as mandatory or excluded (moved to R or E) and appended to $\overrightarrow{S}$.

Example 5.3.1. For example, let us consider $X = \{0,1,2\}$, the variable Sq with initial domain $\langle (), \{0,1,2\}, \emptyset, \emptyset \rangle$ is represented by the tree in Figure 5.3. The intermediate domain $\langle (1), \{0\}, \{1,2\}, \emptyset \rangle$ corresponds to the subtree rooted in (1) (in bold). The valid sequences $\{(1,2), (1,0,2), (1,2,0)\}$ are underlined. The sequences (1) and $(1,0)$ are not valid as they do not contain the required element 2.

The notation $e \in Sq$ where Sq is a sequence variable of domain $\langle \overrightarrow{S}, P, R, E \rangle$, indicates that the element e is required in the sequence ($e \in R$). The prefix sequence variable inherits all the operations defined on the set variable and supports the additional operations summarized in Table 5.1.

Table 5.1: Operations supported by prefix sequence variables

Notation	Operation	Complexity
`isBound(`Sq`)`	return true iff P is empty and $\|S\| = \|R\|$	$\Theta(1)$
`appends(`Sq, e`)`	append e to the end of $\overrightarrow{S}$, moves e to R if needed, fails if e is in E	$\Theta(1)$
`lastAppended(`Sq`)`	return the last element of $\overrightarrow{S}$	$\Theta(1)$
`isMember(`Sq, e`)`	return true iff e is present in $\overrightarrow{S}$	$\Theta(1)$
`allMember(`Sq`)`	enumerate $\overrightarrow{S}$	$\Theta(\|S\|)$
`isAppendable(`Sq, e`)`	return true iff $e \in (R \setminus S) \cup P$	$\Theta(1)$
`allAppendable(`Sq`)`	enumerate $(R \setminus S) \cup P$	$\Theta(\|R \setminus S\| + \|P\|)$

5.3.1 Implementation

The implementation proposed for the Prefix Sequence Variable uses array-based sparse sets as in [de +13] to ensure efficient update and reversibility during a backtracking depth-first-search. It consists of an array of length

$|X|$ called `elems` and two reversible integers: `r` and `p`. The position of the elements of X in `elems` indicates in which subset the element is. Elements before the position `r` are part of R while elements starting from position `p` are part of E. Elements in between are part of P.

Similarly, $\overrightarrow{S}$ is represented by an array of length $|X|$ called `members` and a reversible integer `m`. The `m` first elements in `members` correspond to the sequence $\overrightarrow{S}$. Two additional arrays called `elemPos` and `memberPos` map each element of X with its positions in `elems` and `members`, allowing access in $\Theta(1)$. An illustration of the sparse set representation for the variable Sq with a domain of $\langle (f, b), \{c\}, \{b, e, f\}, \{a, d\}\rangle$ is given in Figure 5.4.

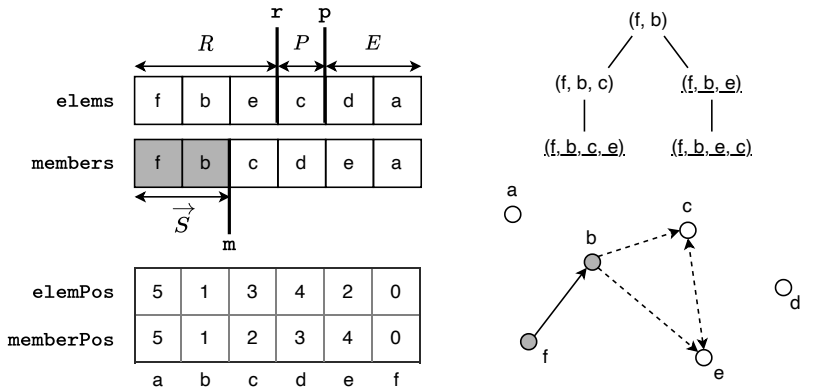

Figure 5.4: The prefix sequence variable domain $\langle (f, b), \{c\}, \{b, e, f\}, \{a, d\}\rangle$ represented using sparse sets (left) and the corresponding tree and paths (right). Valid sequences are underlined in the tree.

Access and identity operations are implemented by comparing the position of an element with the values of `r`, `p` and `m`. Modification operations (`append`, `requires` and `excludes`) are implemented by swapping the element whose status is modified with the element after `m` or `r` or before `p` and incrementing the corresponding reversible integer. Backtracks are done by resetting the reversible integers to their previous value. Enumeration operations are done by iterating over the subpart of the array corresponding to the desired set or sequence. Additional methods allow specifying under which case a propagator should be notified of changes in the domain. Possible events triggering such notifications are: appending, requiring or excluding an element and when the sequence variable is bound.

5.4 Insertion Sequence Variable

Compared to the prefix sequence variable, the insertion sequence variable extends the domain from Definition 16. by adding a set of tuples that contains all possible insertions in $\overrightarrow{S}$.

Definition 17. An insertion sequence variable Sq on a set X is a variable whose domain is represented by a tuple $\langle \overrightarrow{S}, I, P, R, E \rangle$ where $\langle P, R, E \rangle$ is the domain of a set variable on X, $\overrightarrow{S}$ is a sequence $\in \overrightarrow{\mathcal{P}}(R)$ and I is a set of tuples (e, p), each corresponding to a possible insertion. The domain of Sq, also noted $D(Sq)$, is defined as

$$\langle \overrightarrow{S}, I, P, R, E \rangle \equiv \left\{ \overrightarrow{S}' \in \overrightarrow{\mathcal{P}}(P \cup R) \mid R \subseteq S' \wedge \overrightarrow{S} \underset{I}{\overset{*}{\Longrightarrow}} \overrightarrow{S}' \right\} \tag{5.6}$$

As for the prefix sequence variable, Sq is bound if P is empty and $|S| = |R|$. Initially, all elements of the domain are optional ($\in P$). During the search, elements can be set as mandatory or excluded (moved to R or E), inserted in $\overrightarrow{S}$ and possible insertions can be removed from I.

Lemma 1. Checking the consistency of the domain $\langle \overrightarrow{S}, I, P, R, E \rangle$ is NP-complete.

Proof. It requires verifying the following properties:

$$\exists \overrightarrow{S}' \mid \overrightarrow{S} \underset{I}{\overset{*}{\Longrightarrow}} \overrightarrow{S}' \wedge R \subseteq S' \tag{5.7}$$

$$\forall e \in P, \exists S' \mid \overrightarrow{S} \underset{I}{\overset{*}{\Longrightarrow}} \overrightarrow{S}' \wedge R \cup \{e\} \subseteq S' \tag{5.8}$$

The Hamiltonian path problem for a directed graph $G = (\mathcal{V}, \mathcal{E})$ can be reduced to checking the consistency of the domain $D(Sq) = \langle \overrightarrow{S} = (), I = \mathcal{E}_{reverse} \cup \{(v, \alpha) \mid \forall v \in \mathcal{V}\}, P = \emptyset, R = \mathcal{V}, E = \emptyset \rangle$ where $\mathcal{E}_{reverse}$ is the result of applying the *reverse* operation on each edge $(i, j) \in \mathcal{E}$ defined as $(i, j)_{reverse} = (j, i)$. □

Consequently, instead of checking the full domain consistency at each change in the domain, the following invariant is maintained internally by the sequence variable:

$$P \cup R \cup E = X \wedge P \cap R = R \cap E = P \cap E = \phi \tag{5.9}$$

$$S \subseteq R \tag{5.10}$$

$$\forall (e, p) \in I, e \notin S \wedge e \notin E \wedge p \notin E \tag{5.11}$$

$$\forall p \in S, \nexists (e, p) \in I \implies e \in E \tag{5.12}$$

At any moment: $P \cup R \cup E$ form a partition of X (5.9); any member of $\overrightarrow{S}$ is required (5.10); any member of $\overrightarrow{S}$ cannot be inserted in $\overrightarrow{S}$; any excluded element cannot be inserted in $\overrightarrow{S}$ and is not a valid predecessor (5.11); any element that cannot be inserted at any position in $\overrightarrow{S}$ is excluded (5.12). Additionally, elements can only be transferred from P to R or P to E. Note that it is possible for an element to be required (present in R) but not yet part of the internal sequence $\overrightarrow{S}$.

Example 5.4.1. Let us consider $X = \{a, b, c, d, e, f\}$, the variable Sq of domain $\langle \overrightarrow{S} = (f, b), I = \{(c, \alpha), (c, e), (c, f), (e, c), (e, f)\}, P = \{c\}, R = \{b, e, f\}, E = \{a, d\}\rangle$ corresponds to the sequences $\{(f, e, b), (c, f, e, b), (f, c, e, b), (f, e, c, b)\}$. The sequences $\{(f, b), (c, f, b), (f, c, b)\}$ are not valid as they do not contain e which is required.

The insertion sequence variable inherits all the operations defined on the set variable (see Table 2.1) and supports the additional operations summarized in Table 5.2.

Table 5.2: Operations supported by insertion sequence variables

Operation	Description	Complexity
`isBound`(Sq)	return true iff Sq is bound	$\Theta(1)$
`isMember`(Sq, e)	return true iff e is present in $\overrightarrow{S}$	$\Theta(1)$
`allMembers`(Sq)	enumerate $\overrightarrow{S}$	$\Theta(\lvert S\rvert)$
`allCurrentInserts`(Sq)	enumerate $\{(e, p) \in I \mid p \in S\}$	$O(\min(\lvert I\rvert, \lvert S\rvert))$
`nextMember`(Sq, e)	return the successor of e in $\overrightarrow{S}$	$\Theta(1)$
`insert`(Sq, e, p)	insert e in $\overrightarrow{S}$ after p, update P, R and I, fail if $e \in E \vee p \notin S$	$\Theta(1)$
`canInsert`(Sq, e, p)	return true iff $(e, p) \in I$	$\Theta(1)$
`allInserts`(Sq)	enumerate I	$\Theta(\lvert I\rvert)$
`remInsert`(Sq, e, p)	remove (e, p) from I	$\Theta(1)$

5.4.1 Implementation

As for the Prefix Sequence implementation, the internal set variable $\langle P, R, E\rangle$ is implemented using an array-based sparse set called `elems` and two reversible integers: `r` and `p`. The `elemPos` array maps each element of X with its position in `elems`, allowing access in $\Theta(1)$.

The internal partial sequence $\overrightarrow{S}$ is implemented using a reversible chained structure. An array of reversible integers called `succ` indicates for each element its successor in the partial sequence. An element which is not part of the partial sequence points towards itself. An additional dummy element α marks the start and end of the partial sequence. It can be specified as predecessor in the insertion operation to insert an element at the beginning of the sequence or in an empty sequence. Inserting an element e in the par-

tial sequence after p consists in modifying the successor of e to point to the previous successor of p and modifying the successor of p to point to e.

The set of possible insertions I is implemented using an array of sparse sets called `posPreds`. For each element, the corresponding sparse set contains all the possible predecessors after which the element can be inserted. If the element is a member of the sequence $\overrightarrow{S}$ or excluded, its set is empty. The sparse sets are initialized with the following domain: $R \cup P \cup \{\alpha\}$. Constraints may remove possible insertions during their propagation. If doing so results in an empty set, the corresponding element is excluded according to the invariant (5.12).

An illustration of the domain representation for the variable Sq with a domain of $\langle \overrightarrow{S} = (f, b), I = \{(c, \alpha), (c, e), (c, f), (e, c), (e, f)\}, P = \{c\}, R = \{b, e, f\}, E = \{a, d\}\rangle$ is given in Figure 5.5.

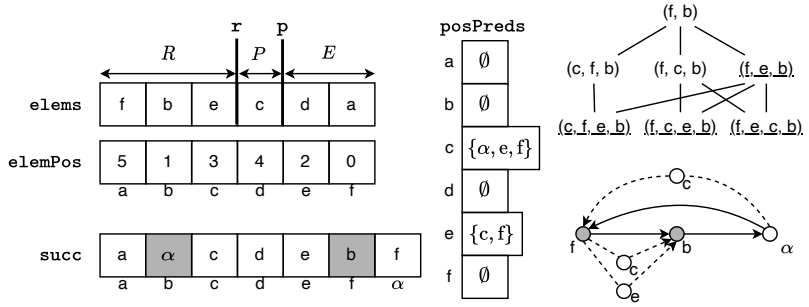

Figure 5.5: The insertion sequence variable domain $\langle \overrightarrow{S} = (f, b), I = \{(c, \alpha), (c, f), (c, e), (e, c), (e, f)\}, P = \{c\}, R = \{b, e, f\}, E = \{a, d\}\rangle$ (left and middle) and the corresponding lattice (with valid sequences underlined) and graphical representation (right)

5.5 Constraints on Sequence Variables

This section presents several constraints that are defined for both implementations of the sequence variable. For most constraints, the propagation algorithm differs depending on the implementation of the variable.

5.5.1 First and Last constraints

The constraint $\texttt{First}(Sq, f)$, holds if the element f is the first in the sequence.

$$\texttt{First}(Sq, f) = \left\{ \overrightarrow{S}' \in D(Sq) \mid \left(\nexists e \in S' \mid e \overset{\vec{S}'}{\prec} f \right) \right\} \tag{5.13}$$

The constraint $\texttt{Last}(Sq, l)$ holds if the element l is the last in the sequence.

$$\texttt{Last}(Sq, l) = \left\{ \overrightarrow{S}' \in D(Sq) \mid \left(\nexists e \in S' \mid l \overset{\vec{s'}}{\prec} e \right) \right\} \tag{5.14}$$

Prefix Propagation In the case of the prefix sequence variable, The propagation for `First` is called only once when the constraint is posted. It appends f as the first element if $\overrightarrow{S}$ is empty. Otherwise, it checks the first element of $\overrightarrow{S}$ and fails if it is different from f. When `Last` is posted, l is made required. The propagation is called each time an element is appended or required. It ensures that l cannot be appended while at least one other element is present in R. When l is appended, all the remaining elements in P are excluded.

Insertion Propagation For the insertion sequence implementation, the propagation algorithm for both constraints is called only once when the constraint is posted. If not already member of the sequence, the element f or l, is inserted as first or last in $\overrightarrow{S}$ (after or before α). Then, I is filtered to remove all insertions before f (insertions with α as predecessor), respectively after l (insertions whit l as predecessor). The constraint fails if f or l is already member of the sequence but not as first or last element (not counting α).

5.5.2 Dependency constraint

The constraint $\texttt{Dependency}(Sq, U)$ ensures that all the elements of U are in the sequence or none of them:

$$\texttt{Dependency}(Sq, U) = \left\{ \overrightarrow{S}' \in D(Sq) \mid U \subseteq S' \vee U \cap S' = \emptyset \right\} \tag{5.15}$$

Propagation The propagation algorithm is the same for both implementations. It is called when an element from U is inserted, required or excluded. If the element of U is inserted or required, all the others are required. Conversely, if the element is excluded, all the others are excluded from the sequence.

Example 5.5.1. Consider the prefix sequence variable Sq with domain $\langle (0), \{1, 2, 4\}, \{0, 3\}, \emptyset \rangle$. The constraint $\texttt{Dependency}(Sq, \{1, 3, 4\})$ reduces the domain to $\langle (0), \{2\}, \{0, 1, 3, 4\}, \emptyset \rangle$ since 3 is already required.

5.5.3 Precedence constraint

This constraint enforces precedence between the elements of a sequence $\overrightarrow{O}$ in the sequence variable:

$$\texttt{Precedence}(Sq, \overrightarrow{O}) = \left\{ \overrightarrow{S}' \in D(Sq) \mid \forall a, b \in S' \cap O, a \overset{\overrightarrow{O}}{\prec} b \implies a \overset{\overrightarrow{S'}}{\prec} b \right\} \tag{5.16}$$

Prefix Propagation The propagation is triggered whenever an element e from O is appended in the variable Sq. All the elements before e in $\overrightarrow{O}$ are excluded in the variable Sq.

Example 5.5.2. Let us consider the sequence variable Sq with domain $\langle (0,2), \{1,3,4\}, \{0,2\}, \emptyset \rangle$. The constraint $\texttt{Precedence}(Sq, \overrightarrow{O})$ where $\overrightarrow{O} = (0,1,2,3,4)$ reduces this domain to $\langle (0,2), \{3,4\}, \{0,2\}, \{1\} \rangle$.

Insertion Propagation The propagation for the insertion sequence uses two different algorithms. The first one is called whenever $\overrightarrow{S}$ is modified. It consists in iterating conjointly over $\overrightarrow{S}$ and $\overrightarrow{O}$ to ensure that elements of $S \cap O$ appear in the same order in $\overrightarrow{S}$ as in $\overrightarrow{O}$. The constraint fails otherwise. Its complexity is linear.

The second propagation algorithm is called when a change is detected in $\overrightarrow{S}$ or I. It filters out from I all insertions that do not respect the order of $\overrightarrow{O}$ based on the current state of $\overrightarrow{S}$ and I. To do so, Algorithm 4 is called twice: a first time with $\overrightarrow{O}$ and a second time with $\overrightarrow{O}$ reversed. First, each member of $\overrightarrow{S}$ is associated to its index in $\overrightarrow{S}$ in a map called `positions` (line 1). Then, the elements of $\overrightarrow{O}$ are iterated over in the specified order. The position of the previous element of $\overrightarrow{O}$ present in the sequence is kept in memory by the variable `precPos` (line 5). Elements of $\overrightarrow{O}$ which are not members of $\overrightarrow{S}$ have their insertions that would violate the constraint filtered out based on the value of `precPos` by the loop at line 7. The complexity of the algorithm is $O(|O| \cdot |S|)$.

Algorithm 4: `PrecFiltering`$(Sq, \overrightarrow{O}, positions)$

Input: $D(Sq) = \langle \overrightarrow{S}, I, P, R, E \rangle$: Sequence variable domain, $\overrightarrow{O}$: sequence of elements (reversed in second call)

```
1 positions ← maps each e ∈ S with index in S⃗ ;
2 precPos ← −1 ; // initialized to |S| when reversed
3 foreach e ∈ O⃗ do
4     if isMember(Sq, e) then
5         precPos ← positions(e) ;
6     else
7         foreach p ∈ S | (e, p) ∈ I ∧ positions(p) < precPos do
              // < changed to ≥ when reversed
8             remInsert(Sq, e, p) ;
```

5.5.4 Sequence Allocation constraint

Given a list of m sequences $(Sq_v)_{v=0,\dots,m-1}$ of domain $\langle \overrightarrow{S}_v, I_v, P_v, R_v, E_v \rangle$ over a set of elements X. Each element i of X is associated with an integer domain variable $x_i \in D_i$ that indicates the sequence where it appears. Note that D_i may contain a dummy value $\perp$ not corresponding to any sequence. The constraint `SequenceAllocation`$((Sq_v), (x_i), X)$ ensures that each element i of X is a member of the sequence indicated by its associated variable x_i.

$$\begin{aligned}\texttt{SequenceAllocation}((Sq_v), (x_i), X) \equiv \\ (\forall i \in X, \forall v \in \{0..m-1\}, i \in R_v \iff x_i = v) \wedge \\ (\forall v, w \in \{0..m-1 | v \neq w\}, R_v \cap R_w = \emptyset)\end{aligned} \tag{5.17}$$

Propagation The propagation which is the same for both implementations is called when an element is appended, required or excluded in one of the sequence variables Sq_v or when the domain of a variable x_i changes. If an element i is required or appended in one of the sequences Sq_v, its x_i variable is fixed to the value v of the sequence and the element is excluded from the other sequences. If an element i is excluded from a sequence Sq_v, the corresponding value v is removed from its x_i variable. If a value v corresponding to a sequence variable Sq_v is removed from the domain of a x_i variable, the element i is excluded from Sq_v. If a variable x_i is fixed to a singleton corresponding to the value v of a sequence variable Sq_v, the element i is required in Sq_v.

5.5.5 Transition Times constraint

In a scheduling context, the elements to sequence correspond to activities performed over time, each associated with a time window and requiring a minimum transition time to move to the next that depends on the pair of consecutive activities. The `Transition Times` constraint links the sequenced elements with their time window to make sure that transition time constraints are satisfied between any two consecutive elements of the sequence. More formally, each element $i \in X$ is associated with an activity defined by a start $start_i$ and a duration variable dur_i. A matrix $trans_{i,j}$, satisfying the triangle inequality, specifies transition times associated to each couple of activities (i, j). The `TransitionTimes` constraint is then defined as

$$\begin{aligned}&\texttt{TransitionTimes}(Sq, [start], [dur], [[trans]]) \equiv \\ &\left\{ \overrightarrow{S}' \in D(Sq) \mid \forall a, b \in S', a \overset{\overrightarrow{S}}{\prec} b \implies start_b \geq start_a + dur_a + trans_{a,b} \right\}\end{aligned} \tag{5.18}$$

5.5.5.1 Prefix Propagation

The propagation is called whenever an element is appended or required or if one of the bounds of a time window changes. The filtering algorithm is split into three parts: *time windows update*, *feasible path checking and filtering* and *append deduction*.

Time window update This filtering algorithm is used to fix the start and duration of the activities already appended in the sequence based on their order in $\overrightarrow{S}$. It is also used to update the time windows of appendable activities (i.e. in the set $(R \setminus S) \cup P$) based on the last appended activity of the sequence. This update is done in linear time.

Feasible Path checking and filtering This algorithm is used to check that there exists at least one feasible extension of the current sequence composed of the required activities not yet appended (i.e. in the set $R \setminus S$) that satisfies the transition time constraints. The function `feasiblePath`$(\ell = lastAppended(Sq), \Omega = R \setminus S, t = start_\ell + dur_\ell, d)$ checks that there exists at least one feasible sequencing of the activities in Ω satisfying the transition time constraints after having completed the activity ℓ at a time t. This problem is NP-Complete [GJ79]. An exact algorithm consists in performing a recursive depth first backtracking search (DFS) enumerating all possible sequences until a feasible one is eventually found.

Algorithm 5 is based on this approach. A pruning is done in the loop at line 6 if one realizes that no activity can be appended. This loop checks that all the activities can possibly be appended as the one following the activity ℓ. By the triangle inequality assumption of the transition times, if at least one activity of Ω cannot be appended directly after ℓ, then it can surely not be appended later in time if some activities were inserted in between. Therefore `false` is returned in such case which corresponds to the infeasibility pruning. Otherwise every possible extension is considered recursively at line 13 and if one possible extension leads to a positive outcome, `true` is returned. Since this algorithm is exponential in $|\Omega|$, the depth of the tree is limited to d. If this limit is reached, the path is assumed to be feasible by returning `true` at line 11.

The time complexity of Algorithm 5 is $O(|\Omega|^d)$ in worse case as it corresponds to a depth-first search of depth d with a branching factor $|\Omega|$. In order to reduce the time complexity of the successive calls to `feasiblePath`, memoization is used to avoid exploring several times a partial extension that can be proven infeasible or feasible based on previous executions. This optimization is implemented with gray lines. A global map called `cache` is assumed to contain keys composed of the arguments of the function, that is

Algorithm 5: `feasiblePath`(ℓ, Ω, t, d)

Input: ℓ: last activity visited, Ω: set of activities to reach, t: departure time from ℓ, d: depth and *cache*: memoization map

```
1  if Ω = ∅ then
2  |  return true;
3  (t_f, t_i) ← cache.getOrElse((ℓ, Ω), (−∞, +∞)) ;
4  if t ≤ t_f then return true;
5  if t ≥ t_i then return false;
6  for a ∈ Ω do
7  |  if max(start_a) < max(t + trans_{ℓ,a}, min(start_a)) then
8  |  |  cache.update((ℓ, Ω), (t_f, min(t_i, t))) ;
9  |  |  return false ; // pruning since infeasible sequence
10 if d ≤ 0 then
11 |  return true ; // pruning since maximum depth reached
12 else
13 |  for a ∈ Ω sorted in increasing (min(start_a) + min(dur_a)) do
14 |  |  if feasiblePath(a, Ω \ {a}, max(t + trans_{ℓ,a}, min(start_a)), d − 1) then
15 |  |  |  cache.update((ℓ, Ω), (max(t_f, t), t_i)) ;
16 |  |  |  return true;
17 |  return false;
```

a pair with (Ω, ℓ). At each key, the map associates a couple of integer values (t_f, t_i) where t_f is the latest known time at which it is possible to depart from ℓ and find a feasible path among the activities of Ω and t_i is the earliest known time at which the departure from ℓ is too late and there exists no feasible path. Line 3 is called to find if a corresponding entry exists in the map. If it is the case, the departure time t is compared to the couple (t_f, t_i) of the map. If $t \leq t_f$, the value *true* is immediately returned. If $t \geq t_i$, *false* is returned. If $t_f < t < t_i$, the algorithm continues its exploration. The cache is updated at lines 8 and 15 depending on the result found.

This checking algorithm can be used in a shaving-like fashion into Algorithm 6. A value is filtered out from the possible set if its requirement made the sequencing infeasible according to the transition times.

Algorithm 6: `TransitionTimesFiltering`$(Sq : \langle \overrightarrow{S}, P, R, E \rangle, d)$

```
1 ℓ ← lastAppended(Sq) ;
2 cache ← map() ; // initializing the memoization map
3 if !feasiblePath(ℓ, R \ S, start_ℓ + dur_ℓ, d) then
4 |  return failure ;
5 forall a ∈ P do
6 |  if !feasiblePath(ℓ, (R \ S) ∪ {a}, min(start_ℓ) + min(dur_ℓ), d) then
7 |  |  excludes(Sq, a);
```

This `TransitionTimesFiltering` algorithm executes in $O(|P| \cdot |R \setminus$

$S|^d$). Notice that the cache is shared and reused along the calls in order to avoid many subtree explorations.

Append deduction The third propagation algorithm detects if a required activity must be appended next and adds it to the sequence. For each required activity i, the algorithm checks that there exists at least one appendable activity e (i.e. $e \in (R \setminus S) \cup P$) that can be visited before while keeping the required activity feasible afterwards. If it is not the case then the required activity is added to the sequence. The pseudocode is given in Algorithm 7. Its time complexity is $O(|R \setminus S| \cdot |(R \setminus S) \cup P|)$.

Algorithm 7: `AppendDeduction`$(Sq : \langle \overrightarrow{S}, P, R, E \rangle)$

```
1  ℓ ← lastAppended(Sq) ;
2  t ← min(start_ℓ) + min(dur_ℓ) ;
3  forall i ∈ R \ S do
4  |  feasible ← false ;
5  |  forall e ∈ allAppendable(Sq) \ {i} do
6  |  |  dep_e ← max(t + trans_{ℓ,e}, min(start_e)) + min(dur_e) ;
7  |  |  feasible ← max(start_r) ≥ dep_e + trans_{e,i} ;
8  |  |  if feasible then break;
9  |  if !feasible then
10 |  |  append(Sq, i) ;
11 |  |  break ;
```

Example 5.5.3. Consider the following example where $X = \{a, b, c, d\}$ is the set of activities. The transition times between activities (a) and the initial time windows (b, columns start and duration) are given in Table 5.3. We consider the sequence variable Sq of domain $\langle (a), \{c\}, \{a, b, d\}, \emptyset \rangle$. The duration of each activity is fixed at 2. Let us apply the propagation of `TransitionTimes` on this example.

1. *Time window update* is applied. The new time bounds are reported in columns start' and dur' of Table 5.3 (b).

2. *Transition Time Filtering* (Algorithm 6) is applied. The search trees for the checker is displayed in Figure 5.6. The search tree for the filter is displayed in Figure 5.7. Failures are denoted with $\times$ and successes with $\surd$. The initial value of the parameter d is 3. The usage of memoization between both searches is highlighted. The domain is updated to $\langle (a), \emptyset, \{a, b, d\}, \{c\} \rangle$ as the filter detects that c must be excluded.

3. *Append deduction* is applied. The domain after propagation is $\langle (a, d), \emptyset, \{a, b, d\}, \{c\}\rangle$ as sequencing b before d is not feasible.

Table 5.3: Propagation for the sequence $Sq = \langle (a), \{c\}, \{a, b, d\}, \emptyset\rangle$.

	a	b	c	d
a	0	5	6	5
b	5	0	5	7
c	6	5	0	5
d	5	7	5	0

(a)

i	start	dur	start'	dur'
a	[0,8]	2	0	2
b	[0,16]	2	[7,16]	2
c	[0,15]	2	[8,15]	2
d	[0,15]	2	[7,15]	2

(b)

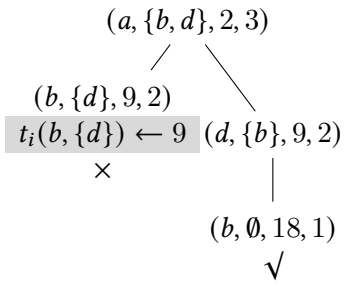

Figure 5.6: Checker search tree for the sequence $Sq = \langle (a), \{c\}, \{a, b, d\}, \emptyset\rangle$.

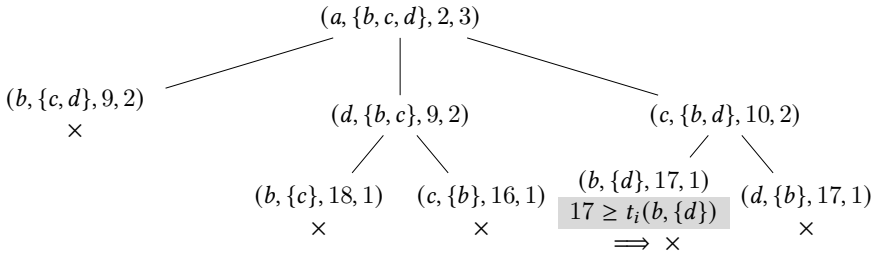

Figure 5.7: Filter search tree for the sequence $Sq = \langle (a), \{c\}, \{a, b, d\}, \emptyset\rangle$.

5.5.5.2 Insertion Propagation

The filtering algorithm is triggered whenever an element is either inserted in $\overrightarrow{S}$ or required or if one of the bounds of a time window changes. The algorithm is split into three parts: *time windows update*, *insertion update* and *feasible path checking and filtering*.

Time window update This filtering algorithm is very similar to the one used for the prefix sequence variable. It is used to adjust the start and duration of the activities already present in $\overrightarrow{S}$. This update is done in linear time by iterating over the elements of the sequence and updating their time windows depending on the time needed to transition from the previous element and to the next element. If the time window of an element is shrunk outside its domain, this leads to a failure.

Insertion update This filtering algorithm is used to filter out the invalid insertions in I based on the current state of $\overrightarrow{S}$ and the transition times of the activities. The algorithm is linear and consists in iterating over I. For each possible insertion, if the transition times between the inserted activity and its predecessor and successor lead to a violation of a time window, the insertion is removed.

Feasible Path checking and filtering We use the same approach as for the prefix sequence variable but adapt it to the insertion sequence variable. The problem of verifying that there exists at least one *transition time feasible* extension of the current sequence composed of the required activities not yet inserted is NP-Complete [GJ79] by a reduction from the TSP. Algorithm 8 is a recursive depth first search used to check that there exists at least one feasible extension of the current sequence composed of the required activities not yet inserted (i.e. in the set $R \setminus S$). Given the current sequence $\overrightarrow{S}$, the recursive call `feasiblePath`(ℓ, p, Ω, t, d) checks that it is possible to build a sequence starting from ℓ at time t that contains at least d elements of Ω and is a super-sequence of the sub-sequence of $\overrightarrow{S}$ starting in p. The parameter ℓ indicates the last element visited at time t whereas the parameter p indicates the last element of S that has been visited (possibly several steps before ℓ). The initial call `feasiblePath`$(\ell = \alpha, p = \alpha, \Omega = R \setminus S, t = 0, d)$ thus checks that there exists a super-sequence of $\overrightarrow{S}$ containing d elements of $R \setminus S$.

At each node, the algorithm either explores the insertion of a new element after ℓ which corresponds to branching over an element of Ω (line 16) or follows the current sequence $\overrightarrow{S}$ which consists in branching over the successor of p in $\overrightarrow{S}$ (line 20). A pruning is done at lines 2 and 9 if one realizes that at least one activity cannot be reached. By the triangle inequality assumption of the transition times, if either the successor of p or at least one activity of Ω cannot be reached directly after ℓ, then it can surely not be reached later in time if some activities were visited in between. Therefore `false` is returned in such case which corresponds to the infeasibility pruning. The possible extensions considered recursively at line 16 are based on the current state of I and the value of p. The maximum depth is controlled by the parameter d to

Algorithm 8: `feasiblePath`(ℓ, p, Ω, t, d)

Input: ℓ: last element reached, p: previous element reached in $\overrightarrow{S}$, Ω: set of elements to reach, t: departure time from ℓ, d: depth, $Sq = \langle \overrightarrow{S}, I, P, R, E \rangle$: Sequence variable, cache: memoization map

1 $n \leftarrow$ `nextMember`(Sq, p) ;
2 **if** $n \neq \alpha$ *and* $t + trans_{\ell,n} > \max(start_n)$ **then**
3 | **return** false;
4 **if** $\Omega = \emptyset$ **then**
5 | **return** true;
6 $(t_f, t_i) \leftarrow$ cache.`getOrElse`$((\ell, p, \Omega), (-\infty, +\infty))$;
7 **if** $t \leq t_f$ **then return** true;
8 **if** $t \geq t_i$ **then return** false;
9 **for** $a \in \Omega$ **do**
10 | **if** $t + trans_{\ell,a} > \max(start_a)$ **then**
11 | | cache.`update`$((\ell, p, \Omega), (t_f, \min(t_i, t)))$;
12 | | **return** false ; `// pruning (infeasible sequence)`
13 **if** $d \leq 0$ **then**
14 | **return** true ; `// pruning (maximum depth reached)`
15 **else**
16 | **for** $a \in \Omega \mid (a, \ell) \in I$, *sorted in increasing* $(\min(start_a) + \min(dur_a))$ **do**
17 | | **if** `feasiblePath`$(a, p, \Omega \setminus \{a\}, \max(t + trans_{\ell,a}, \min(start_a) + \min(dur_a)), d - 1)$ **then**
18 | | | cache.`update`$((\ell, p, \Omega), (\max(t_f, t), t_i))$;
19 | | | **return** true;
20 | **if** $n \neq \alpha$ *and* `feasiblePath`$(n, n, \Omega, \max(t + trans_{\ell,n}, \min(start_n) + \min(dur_a)), d)$ **then**
21 | | cache.`update`$((\ell, p, \Omega), (\max(t_f, t), t_i))$;
22 | | **return** true;
23 | **return** false;

avoid prohibitive computation. The algorithm can thus return a false positive result by returning `true` at line 14 if this limit is reached.

The time complexity of Algorithm 8 is $O(|S| \cdot |\Omega|^d)$ in worse case as it corresponds to an iteration over $\overrightarrow{S}$ with a depth-first search of depth d and branching factor $|\Omega|$ at each step. In practice, as the branching is based on I, the search tree will often be smaller. In order to reduce the time complexity of the successive calls to `feasiblePath`, a cache is used to avoid exploring several times a partial extension that can be proven infeasible or feasible based on previous executions. A global map called `cache` is assumed to contain keys composed of the arguments of the function, that is a tuple with (Ω, ℓ, p). At each key, the map associates a couple of integer values (t_f, t_i) where t_f is the latest known time at which it is possible to depart from ℓ and find a feasible path among the sub-sequence starting after p and the activities of Ω and t_i is the earliest known time at which the departure from ℓ is too late and there exists no feasible path. Line 6 is called to find if a corresponding entry

exists in the map. If it is the case, the departure time t is compared to the couple (t_f, t_i) of the map. If $t \leq t_f$, the value *true* is immediately returned. If $t \geq t_i$, *false* is returned. If $t_f < t < t_i$, the algorithm continues its exploration. The cache is updated at lines 11, 18 and 21 depending on the result found. Usage of the cache is highlighted in gray in Algorithm 8.

This checking algorithm can be used in a shaving-like fashion into Algorithm 9. A value is filtered out from the possible set if its requirement made the sequencing infeasible according to the transition times. This `TransitionTimesFiltering` algorithm executes in $O(|P| \cdot (|S| \cdot |R \setminus S|)^d)$. Notice that the cache is shared and reused along the calls in order to avoid many subtree explorations. Due to the extensive nature of the algorithm, a parameter ρ defines a threshold for the size of P above which the `feasiblePath` algorithm is not executed for each element of P (line 4).

Algorithm 9: `TransitionTimesFiltering`(Sq, d, ρ)

Input: d: maximum depth, ρ: filtering threshold, $Sq = \langle \overrightarrow{S}, I, P, R, E \rangle$: seq. variable

```
1 cache ← map() ; // initializing memoization map
2 if !feasiblePath(α, α, R \ S, 0, d) then
3 |   return failure ;
4 if |P| ≤ ρ then
5 |   forall a ∈ P do
6 |   |   if !feasiblePath(α, α, (R \ S) ∪ {a}, 0, d) then
7 |   |   |   excludes(Sq, a);
```

Example 5.5.4. Let us consider the following example where $X = \{a, b, c, d\}$ is the set of activities. The transition times between activities (a) and the initial time windows (b, column start) are given in Table 5.4. We consider the sequence variable Sq of domain $\langle \overrightarrow{S} = (a, d), I = \{(b, a), (b, d), (c, \alpha), (c, d)\}, P = \{c\}, R = \{a, b, d\}, E = \emptyset \rangle$. The duration of each activity is fixed at 2. Let us apply the propagation of `TransitionTimes` on this example:

1. *Time window update* is applied. The time windows of a and d are reduced. The updated time windows are displayed in Table 5.4 (b, column start').

2. *Insertion update* is applied. The insertion (b, a) is removed from I as b cannot be inserted after a without violation (b would end at the earliest at 9 which implies that d would start at the earliest at 16, outside its time window).

3. *Transition Time Filtering* (Algorithm 9) is applied. The search trees for the checker is displayed in Figure 5.8. The search tree for the filter is displayed in Figure 5.9. Failures are denoted with $\times$ and successes with

$\surd$. The initial value of the parameter d is 3. The domain of Sq after propagation is $\langle (a,d), \{(b,d)\}, \emptyset, \{a,b,d\}, \{c\} \rangle$ as the filter excludes c.

Table 5.4: Propagation on $Sq = \langle (a,d), \{(b,a),(b,d),(c,\alpha),(c,d)\}, \{c\}, \{a,b,d\}, \emptyset \rangle$

	a	b	c	d
a	0	5	6	5
b	5	0	5	7
c	6	5	0	5
d	5	7	5	0

(a)

	start	start'
a	[0,10]	**[0,8]**
b	[0,16]	[0,16]
c	[0,15]	[0,15]
d	[0,15]	**[7,15]**

(b)

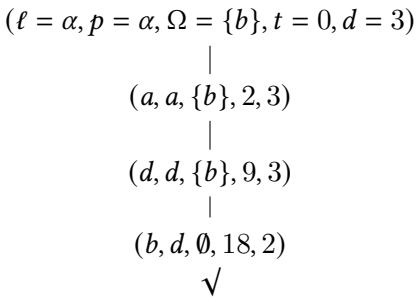

Figure 5.8: Checker search tree for $Sq = \langle (a,d), \{(b,a),(b,d),(c,\alpha),(c,d)\}, \{c\}, \{a,b,d\}, \emptyset \rangle$

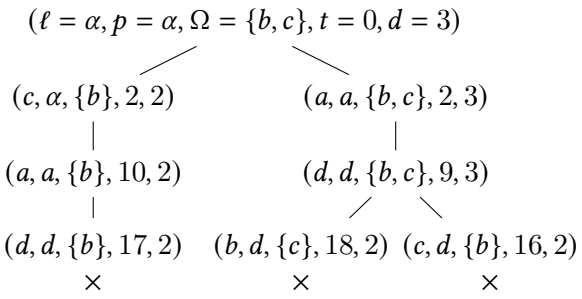

Figure 5.9: Filter search tree for $Sq = \langle (a,d), \{(b,a),(b,d),(c,\alpha),(c,d)\}, \{c\}, \{a,b,d\}, \emptyset \rangle$

5.5.6 Cumulative constraint

In both the DARP and PTP, one has to satisfy requests that correspond to embarking and disembarking a person in a vehicle. The activities of transport

are modeled as pairs of elements in an insertion sequence variable that must occur in this specific order: embarking before disembarking. Also this pair of elements must both be present or absent from the sequence. During the trip, the person occupies some load in the vehicle. By analogy to scheduling problems, a request is called an activity A_i and is composed of the two elements ($start_i$,end_i) corresponding to the embarking and disembarking. This activity will consume a load $load_i$ while it is on the board of the vehicle. The set of activities is denoted A. Also by analogy to scheduling [AB93], we call `Cumulative` the constraint that ensures that the capacity K of the resource is respected at any point in the ordering defined by the sequence Sq over X where $\forall i \in A, start_i, end_i \in X$. More formally

$$\texttt{Cumulative}(Sq, [start], [end], [load], K) \equiv \left\{ \overrightarrow{S}' \in D(Sq) \mid \forall e \in S', \sum_{i \in A | start_i \leq e \leq end_i} load_i \leq K \right\}. \quad (5.19)$$

5.5.6.1 Propagation

The propagation is triggered when new elements are inserted in $\overrightarrow{S}$. It consists in filtering insertions in the current sequence $\overrightarrow{S}$ by checking if they are supported. An insertion for the element corresponding to one extremity of an activity is supported if there exists at least one possible insertion for the other extremity of the activity such that the activity load does not overloads the capacity between both insertion positions.

The first step of the propagation algorithm is to build a minimum load profile that maps each element e of the sequence to the minimal load at this point in the sequence based on the activities that are part of $\overrightarrow{S}$. These can be either fully inserted (both the start and end of the activity $\in \overrightarrow{S}$) or partially inserted (only the start or end $\in \overrightarrow{S}$). For partially inserted activities, the position for the element not yet inserted is chosen among the possible insertions in I as the closest one to the inserted element. Note that a violation of the capacity at this point would trigger a failure.

Once the cumulative profile is built, possible insertions for activities that are partially inserted are filtered. The algorithm used consists in iterating over $\overrightarrow{S}$ starting from the inserted element. Possible insertions for the missing element are considered and allowed as long as the load of the activity can be added to the minimal load profile without overloading the capacity. If the capacity is overloaded at some point, the current insertion as well as the insertions not yet reached are removed.

Finally, Algorithm 18 is used to check activities for which neither element is inserted. The loop at line 5 iterates over $\overrightarrow{S}$ starting from the dummy ele-

ment α. When a potential start predecessor is encountered, it is added to the `activeStarts` set which maintains potential valid predecessors for the start element that have been encountered so far (line 7). The boolean `canClose` indicates if there exists at least one possible insertion position for the start of the activity that would be valid if the end is inserted at this point. It is set to true whenever a start predecessor is added to `activeStarts`. If adding the activity to the load profile for the current element violates the capacity, `canClose` is set to false and `activeStarts` is emptied as the potential start predecessors will not be matched to a valid insertion for the end element. When a valid predecessor for the end element is encountered, the end predecessor and all the start predecessors in `activeStarts` are validated (lines 13 and 14). The possible predecessors that have not been validated at the end of the loop are removed at line 18.

Algorithm 10: `CumulFiltering`$(Sq, start, end, load, K, profile)$

Input: $start, end, load$: starts, ends and loads of activities, C: capacity, $Sq = \langle \overrightarrow{S}, I, P, R, E \rangle$: Sequence variable, $profile$: minimum load profile

1 **forall** $i \mid start_i \notin S \wedge end_i \notin S$ **do**
2 $activeStarts \leftarrow \emptyset$;
3 $current \leftarrow \alpha$;
4 $canClose \leftarrow false$;
5 **do**
6 **if** $canInsert(Sq, start_i, current)$ **then**
7 $activeStarts \leftarrow activeStarts \cup \{current\}$;
8 $canClose \leftarrow true$;
9 **if** $profile(current) + load_i > K$ **then**
10 $activeStarts \leftarrow \emptyset$;
11 $canClose \leftarrow false$;
12 **if** $canInsert(Sq, end_i, current) \wedge canClose$ **then**
13 $current$ is valid predecessor for end_i ;
14 $\forall p \in activeStarts$, p is valid predecessor for $start_i$;
15 $activeStarts \leftarrow \emptyset$;
16 $current \leftarrow$`nextMember`$(Sq, current)$
17 **while** $current \neq \alpha$;
18 remove predecessors for $start_i$ and end_i that have not been validated ;

The complexity to build the minimum load profile is linear. The complexity to check all the activities $\in A$ is $O(|A| \cdot |S|)$.

Example 5.5.5. Let us consider four activities: $A_0 = [a, e], A_1 = [b, f], A_2 = [c, g]$ and $A_3 = [d, h]$. Each activity A_i has a load of 1. The capacity is $K = 3$. The current partial sequence is $\overrightarrow{S} = (a, b, c, e, f)$. Before propagation, the current possible insertions in I are: $\{(d, \alpha), (d, a), (d, b), (d, g), (g, d), (g, e), (g, f), (g, h), (h, a), (h, c), (h, e), (h, d), (h, g)\}$. Note that the possible insertions that are not in the current sequence $((d, g), (g, d), (g, h), (h, d), (h, g))$ will be ig-

nored by the filtering algorithm. Let us propagate the `Cumulative` constraint:

1. The minimal load profile is built based on $A_0 = [a, e]$ and $A_1 = [b, f]$ which are both fully inserted and $A_2 = [c, g]$ which is partially inserted (only c is member in $\overrightarrow{S}$). The possible insertion for the end of A_2 (g) that is the closest to its start (c) is (g, e). Thus, A_2 is considered ending after e to compute the minimum load profile which is $\{\alpha : 0, a : 1, b : 2, c : 3, e : 2, f : 0\}$.

2. The possible insertions for the partially inserted activity A_2 are filtered. The sequence is iterated over starting from c. As (g, e) is part of the minimal load profile, it is validated. The remaining possible insertion (g, f) is reached without overloading the capacity and thus validated.

3. The possible insertions for non-inserted activity $A_3 = [d, h]$ are filtered. To do so, Algorithm 18 iterates over the elements in $\overrightarrow{S}$, starting from α. Both α and a are added to `activeStart` and `canClose` is set to *true*. When considering a as possible predecessor for h, as `canClose` is true, the insertions (h, a), (d, α) and (d, a) are validated. Afterwards b is added to `activeStart`. When considering c, adding the activity A_3 at this point would overload the capacity K. Thus, `canClose` is set to *false* and `activeStart` is emptied. c and d are not validated as possible predecessors, as `canClose` is *false* when they are considered.

The load profile and possible insertion positions for d and h are illustrated in figure 5.10.

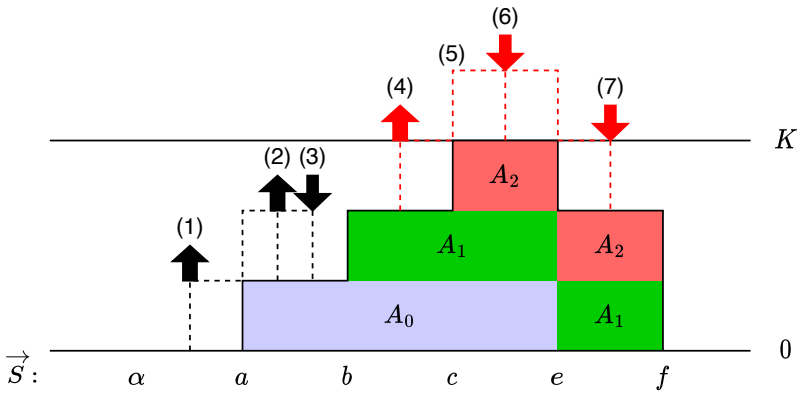

Figure 5.10: Minimum load profile (in gray) and possible insertions for d (up arrows) and h (bottom arrows). Invalid insertions are displayed in red.

At the end of the propagation, the validated insertions are (g, e), (g, f), (d, α), (d, a) and (h, a). The possible insertions (d, b), (h, c) and (h, e) are removed from I.

5.5.7 Max Distance constraint

In routing contexts the sequence variable may be used to represent the route of a vehicle. In this case, elements of the sequence corresponds to steps on the route, which are separated by transitions in order for the vehicle to move from one to another. The `MaxDistance` constraint ensures that the total length of the route is equal to an integer variable.

Given a matrix $trans_{i,j}$ that specifies a transition distance associated to each couple of elements (i, j) and an integer variable D, the `MaxDistance` constraint is defined as

$$\texttt{MaxDistance}(Sq, D, trans_{i,j}) = \left\{ \overrightarrow{S}' \in D(Sq) \mid \sum_{(a,b) \mid a \xrightarrow{\overrightarrow{S}'} b} trans_{a,b} \leq D \right\} \tag{5.20}$$

Propagation The propagation is nearly the same for both implementations. It uses three algorithms. The first one, which is called when D or $\overrightarrow{S}$ is modified, computes the length of the current internal sequence $\overrightarrow{S}$ and updates the lower bound of D. The second algorithm is used only for the insertion sequence variable. It is called when D or I is modified. It filters out insertions in I that would result in a sequence longer than the upper bound of D. The last propagation algorithm is called whenever a new element is added to R or inserted in $\overrightarrow{S}$.

It consists computing a minimum spanning tree over the graph formed by the path of $\overrightarrow{S}$ and all the edges connecting the non-inserted required elements to the elements of $\overrightarrow{S}$. The Kruskal [Kru56] algorithm is used to compute the MST. The length of the MST is then used as lower bound for D. The complexity of the two first algorithms is linear. The last algorithm has a complexity of $O(E \cdot \log(E))$ where $E = |R \setminus S|^2 + |R \setminus S| \cdot |S| + |S|$.

5.6 Applications of the Sequence Variable

This section presents the application of the prefix and insertion sequence variables on the PTP. Additional experiments were performed with the insertion sequence variable on the DARP.

5.6.1 Patient Transportation Problem (PTP)

The first problem considered is the Patient Transportation Problem (PTP). Its formal definition is given in Section 4.1.1.

5.6.1.1 Model with Prefix Sequence Variables

The model proposed for the PTP associates each stop of a vehicle at a specific location to an activity. Each request $r \in R$ consists in one or two travels. The set of all travels is noted T. Each travel $t \in T$ is composed of a couple of stops (p, d) where p corresponds to the pickup of the patient and d to its drop off. The notation $i \in r$ indicates that the stop i is part of the request r. The set of all stops is noted X. The transition time between the locations of two stop a and b is noted $trans_{a,b}$. For each vehicle $v \in V$, two additional stops sd_v and ed_v are used to model the visit at the start ($start_v$) and end (end_v) of the depots.

The variables of the model are the following:

- For each vehicle $v \in \mathcal{V}$, a prefix sequence variable Sq_v represents the vehicle route. Its initial domain is $\langle (), X, \emptyset, \emptyset \rangle$.

- Each request $r \in R$ is associated with a boolean variable Rs_r corresponding to 1 if the request r is selected (realized in the solution) and 0 otherwise.

- Each stop $i \in X$ is associated with an activity defined by a triplet of variables (s_i, d_i, e_i) where s_i is the starting time of the activity, d_i is the duration of the activity e_i is the ending time of the activity. The relation $e_i = s_i + d_i$ must hold. The initial domain of these variables is defined by the corresponding request $r \in R$ and whether the stop is part of a forward or a backward trip: for a forward trip, $s_i \in [rdv_r - maxw_r, rdv_r - srv_r]$ and $e_i \in [rdv_r - maxw_r + svr_r, rdv_r]$; for a backward trip, $s_i \in [rdv_r + drdv_r, rdv_r + drdv_r + maxw_r - srv_r]$ and $e_i \in [rdv_r + drdv_r + srv_r, rdv_r + maxw_r]$. in both cases, $d_i = srv_r$. A variable x_i indicates which vehicle visits the stop. Its initial domain contains all the compatible vehicle indices ($cat_r \in C_v$) along with an additional value $\perp$ which indicates that the stop is not visited by any vehicle. Variables for the depot stops of a vehicle v are initialized with a vehicle variable $x_i = v$ and time windows corresponding to the availability of the vehicle: $s_i \in [savail_v, eavail_v]$, $e_i \in [savail_v, eavail_v]$, $d_i = e_i - s_i$.

- Alternative stop activities are duplicated for each vehicle and defined by the variables $(salt_{i,v}, dalt_{i,v}, ealt_{i,v})_{i \in X, v \in V}$. They are linked with the

original stop activities by an `element` constraint [VC88]. These auxiliary variables are used for the `TransitionTimes` constraint over the sequence variable corresponding to their vehicle.

- For each travel $t = (p, d) \in T$, an activity is defined by the variables (st_t, dt_t, et_t) with $st_t = s_p$, $et_t = e_d$ and $dt_t = et_t - st_t$. The vehicle variable x_t of the travel t is the vehicle variable of its pickup stop $x_t = x_p$. The variable $lt_t = ld_r$ indicates the load of the associated request r. These variables are used for the cumulative global constraint.

The full model consists in:

$$\max(\sum_{r \in R} Rs_r) \tag{0}$$

s.t.

$$\texttt{First}(Sq_v, sd_v) \qquad \forall v \in V \tag{1}$$

$$\texttt{Last}(Sq_v, ed_v) \qquad \forall v \in V \tag{2}$$

$$\texttt{Cumulative}((st_t), (dt_t), (et_t), lt_t, (x_t), k_v, v) \qquad \forall v \in V \tag{3}$$

$$\texttt{TransitionTimes}(Sq_v, (salt_{i,v}), (dalt_{i,v}), trans) \qquad \forall v \in V \tag{4}$$

$$\texttt{Precedence}(Sq_v, (p, d)) \qquad \forall v \in V, t = (p, d) \in T \tag{6}$$

$$\texttt{Dependency}(Sq_v, (p, d)) \qquad \forall v \in V, t = (p, d) \in T \tag{7}$$

$$Rs_r \equiv (x_i \neq \perp) \qquad \forall r \in R, i \in X \mid i \in r \tag{12}$$

$$\texttt{element}(x_i, (salt_{i,v}), s_i)$$

$$\texttt{element}(x_i, (dalt_{i,v}), d_i)$$

$$\texttt{element}(x_i, (ealt_{i,v}), e_i) \qquad \forall i \in X \tag{13}$$

$$s_d \geq e_p + trans_{p,d} \qquad \forall t = (p, d) \in T \tag{14}$$

$$\texttt{SequenceAllocation}((Sq_v), (x_i), X) \tag{15}$$

Numbering refers to the formal definition in Section 4.1.1 when applicable. The objective function (0) maximizes the number of served requests. Constraint (1) ensures that each vehicle leaves from its starting depot while Constraint (2) ensures that each vehicle will end at its end depot. Constraint (3) ensures that the number of places occupied by patients in a same vehicle v cannot exceed its capacity k_v. This constraint is referred in the literature as the cumulative resource global constraint [AB93] (see Section 2.2.1.2). We use the filtering algorithm of Gay et al. [GHS15a]. Constraint (4) ensures the consistency of the time windows regarding the transition times between stops. Constraints (5) and (8) to (11) from the formal definition are enforced through the initial domain of the corresponding variables. Constraint (6) ensures that each pickup stop is visited before its corresponding drop stop. Constraint (7) ensures that each couple of pickup and drop stops is present in a same vehicle. Constraint (12) links the request selection variables with the vehicle variables of the request travel(s). The group of `element` constraints (13) links each stop activity with the alternative activity corresponding to the vehicle performing the activity. Constraint (14) is an additional constraint that

improves the propagation by ensuring that the transition time is taken into account in the time window of the stops of a same travel. Constraint (15) enforces that the vehicle variable of each stop corresponds to the sequence containing the stop or the value $\perp$ if the stop is not visited.

5.6.1.2 Model with Insertion Sequence Variable

The model with the insertion sequence variable is similar to the one for the prefix sequence variable with the exception of the following changes:

- For each vehicle $v \in V$, an insertion sequence variable is used instead of a prefix sequence variable to model the vehicle route Sq_v.
- Constraint (3) is enforced by the cumulative constraint defined for the insertion sequence variable in section 5.5.6:

$$\texttt{Cumulative}(Sq_v, (p_r), (d_r), (lr_r), k_v) \forall v \in V \tag{3}$$

5.6.1.3 Search

A Large Neighborhood Search (LNS) [Sha98] is used. The relaxation is inspired by the partial order schedule relaxation of [GLN05]. It consists in randomly selecting a subset of requests. Those can be reinserted anywhere in any sequence while the others are forced to be reassigned in the same sequence in the same linear order. For the prefix sequence variable, the requests that are not selected have their stops assigned (but not appended) to the vehicle that they were part of in the current solution. Precedence constraints ensure that these stops keep the same order as in the current solution. For the insertion sequence variable, the relaxation is easier to perform. It simply consists in inserting the stops in their current sequence in the same order. If the search tree is completely explored during a given number of consecutive iterations given by a stagnation threshold s, the relaxation size is increased.

The search used for the prefix sequence variable is called *sequence selection search*. This heuristic, shown in Algorithm 8, selects the sequence that finishes the earliest (according to the earliest end time of its last appended activity) and consider all its possible extensions. Branches are explored from left to right according to the latest end time of appendable stops. The travel distance between the last appended stop and the new stop is used as a tie breaker.

For the insertion sequence variable, two different search heuristics are considered:

1. **A generic First Fail search**. Similarly as in [JV11], at each step of the search, it selects the element (the stop) not yet decided with the minimal number of possible insertions in all compatible sequences. Then, it branches in a random order over the possible insertions for the element.

Algorithm 11: `SequenceSelection`

```
1 extendableSeqs ← {v | v ∈ 𝒱 ∧ !isBound(Sq_v)} ;
2 if extendableSeqs = ∅ then
3 |   Solution found ;
4 else
5 |   v_min ← argmin_{v|v∈extendableSeqs} min(e_{lastAppended(Sq_v)});
6 |   ℓ ← lastAppended(Sq_{v_min}) ;
7 |   forall i ∈ allAppendable(Sq_{v_min}) by order of (max(e_i), trans_{ℓ,i}) do
8 |   |   branch(append(Sq_{v_min}, i)) ;
```

2. **A problem specific heuristic** called *Slack Driven Search* (SDS). It uses a similar approach to the first fail heuristic to select a stop with a minimal number of possible insertions. The heuristic is guided by a *slack difference* metric which is defined as the total size difference of the time windows of the predecessor and successor of the stop to insert before and after insertion. The intuition is to minimize this difference in order to keep the sequences as flexible as possible and maximizing potential future insertions. Additionally, the branching decisions, each corresponding to a possible insertion for the stop selected, are explored by increasing order of slack difference.

5.6.2 Dial-a-Ride Problem

After the experiments performed on the PTP, additional experiments were done on the DARP. The variant of the DARP considered is the one proposed in [CL03b]. It is described in Section 2.1.5. The prefix sequence variable was not considered for these experiments as it is consistently outperformed by the insertion sequence variable in the PTP experiments and showed poor results when not provided a initial solution.

5.6.2.1 Model with Insertion Sequence Variables

The model is similar to the one used for the PTP (see Sections 5.6.1.2 and 5.6.1.1). For each vehicle $v \in V$, an insertion sequence variable Sq_v represents the vehicle route. Its initial domain is $\langle S = (), P = X, R = \emptyset, E = \emptyset \rangle$. An associated variable $Dist_v$ indicates the total distance of the route.

Each request stop $i \in X$ is associated with an activity defined by a triplet of variables (s_i, d_i, e_i) where s_i is the starting time of the activity, d_i is the duration of the activity e_i is the ending time of the activity. The relation $e_i = s_i + d_i$ must hold. The initial domain of these variables is defined by the time window and service duration of the stop. A variable x_i indicates which vehicle visits the stop. Its initial domain contains all vehicles. Two additional stop activities sd_v and ed_v are added for each vehicle v to model the departure

and end stops of the vehicle at the depot. They are associated with a vehicle variable $x_i = v$.

Alternative stop activities are duplicated for each vehicle and defined by the variables $(salt_{i,v}, dalt_{i,v}, ealt_{i,v})_{i \in X, v \in V}$. They are linked with the original stop activities by an `element` constraint [VC88]. These auxiliary variables are used for the `Transition Times` and `Max Distance` constraints over the sequence variable corresponding to their vehicle.

Each request $r \in R$ is also associated to a start sr_r, an end er_r, a duration dr_r and a vehicle xr_r. Note that these variables are views on the corresponding stop variables: the start of the request corresponds to the start of its pickup stop $sr_r = s_{p_r}$; the end of the request corresponds to the end of its drop stop $er_r = e_{d_r}$; the duration is the difference between the two previous variables $dr_r = er_r - sr_r$ and the vehicle is the one that serves the pickup stop $xr_r = x_{p_r}$. Additionally, a load variable lr_r is created for each request. It has a single value which corresponds to the load of the pickup stop and is used for the cumulative constraint.

The full model is given here:

$$\min\left(\sum_{v \in V} Dist_v\right) \tag{0}$$

s.t.

$$\texttt{First}(Sq_v, sd_v) \qquad \forall v \in V \tag{1}$$

$$\texttt{Last}(Sq_v, ed_v) \qquad \forall v \in V \tag{2}$$

$$\texttt{Cumulative}(Sq_v, (p_r), (d_r), (lr_r), k_v) \qquad \forall v \in V \tag{3}$$

$$\texttt{TransitionTimes}(Sq_v, (salt_{i,v}), (dalt_{i,v}), trans) \qquad \forall v \in V \tag{4}$$

$$\texttt{Precedence}(Sq_v, (p, d)) \qquad \forall v \in V, r = (p, d) \in R \tag{6}$$

$$\texttt{Dependency}(Sq_v, (p, d)) \qquad \forall v \in V, r = (p, d) \in R \tag{7}$$

$$s_d - e_p \leq R_{max} \qquad \forall r = (p, d) \in R \tag{9}$$

$$s_{ed_v} - e_{sd_v} \leq D_{max} \qquad \forall v \in V \tag{10}$$

$$\texttt{SequenceAllocation}((Sq_v), (x_i), X) \tag{11}$$

$$\texttt{MaxDistance}(Sq_v, Dist_v, trans) \qquad \forall v \in V \tag{12}$$

$$\texttt{element}(x_i, (salt_{i,v}), s_i)$$

$$\texttt{element}(x_i, (dalt_{i,v}), d_i)$$

$$\texttt{element}(x_i, (ealt_{i,v}), e_i) \qquad \forall i \in X \tag{13}$$

$$s_d \geq e_p + trans_{p,d} \qquad \forall t = (p, d) \in T \tag{14}$$

Numbering refers to the formal definition in Section 2.1.5 when applicable. Constraint (3) is enforced by a `MaxCumulativeResource` constraint (see Section 2.2.1.2) with the propagator of [GHS15a]. Constraints (5) and (8) from the formal definition are enforced through the initial domain of the corresponding variables. Constraint (12) links the route distance variables and sequence variables between them. The group of `element` constraints (13) links each stop activity with the alternative activity corresponding to the ve-

hicle performing the activity. Constraint (14) is an additional constraint that improves the propagation by ensuring that the transition time is taken into account in the time window of the stops of a same travel.

5.6.2.2 Search

Such as for the PTP, a LNS is used and two search heuristics are considered:

1. The same **generic First Fail heuristic** as the one described in 5.6.1.3;

2. **A problem specific heuristic** called `Cost Driven Search` (CDS). It is similar to the slack driven search presented in Section 5.6.1.3. The cost metric used in [JV11] for their LNS-FFPA algorithm is used to improve the heuristic. The minimum cost between all possible insertions for a stop is used as a tie breaker for the selection of the next stop to insert. Additionally, the branching decisions, each corresponding to a possible insertion for the stop selected, are explored by increasing order of cost.

5.7 Experimental Results

This section reports the comparison of the models presented in section 5.6 with state-of-the-art CP approaches for the PTP and DARP. The models based on prefix sequences variables are referred as the *Insertion Sequence* (PSEQ) approaches and those based on insertion sequences variables are referred as the *Insertion Sequence* (ISEQ) approaches. The generic *First Fail* heuristic is referred as FF. The *Slack Driven* and *Cost Driven* heuristics are referred as SDS and CDS respectively.

Experiments were done on the PTP with a Large Neighborhood Search (LNS) and a Depth First Search (DFS). Both sequence based models are compared with:

1. the model proposed in Section 4.3, referred as *Scheduling with Optional Decision Search* (SCHED+ODS);

2. the model proposed in [LAB18], referred as *Liu CP Optimizer model* (LIU_CPO).

For the experiments in the LNS setting, the same experimental approach as in Section 4.6 is used. A greedy method referred as (GREEDY) is used to compute the initial PTP solutions given to the compared models in the LNS setting. Tests are performed on the benchmark of instances used in Section 4.6. It contains both real exploitation instances and randomly generated instances which are available at [Tho+]. The LNS uses an initial relaxation of

20% of the requests, a failure limit of 500, a stagnation threshold of 50 and an increase factor of 20%. The DFS uses the same heuristics as in the LNS setting. No initial solution is given in this setting. The parameters of the `TransitionTimes` constraint used for both settings are: a maximum depth (if applicable) of 3 and a filtering threshold of 10.

Each approach was run 10 times on each instance, with a time limit of 600 sec. The system used for the experiments is a PowerEdge R630 server (128GB, 2 proc. Intel E5264 6c/12t) running on Linux. The approaches using CP Optimizer were implemented using the Java API of CPLEX Optimization Studio V12.8 [Lab+18]. The other models were implemented on OscaR [Osc12] running on Scala 2.12.4.

In order to compare the anytime behavior of the approaches, we define the *relative distance* of an approach at a time t as the current distance from the best known objective (BKO) divided by the distance to the worse initial objective (WSO):

$$\frac{objective(t) - BKO}{WSO - BKO} \tag{5.21}$$

. If an approach has not found an initial solution, the worse initial objective (WSO) is used as objective value. A relative distance of 1 thus indicates that the approach has not found an initial solution or is stuck at the initial solution while a relative distance of 0 indicates that the best known solution has been reached.

5.7.1 PTP with LNS

The results of the experiments on the PTP instances in a LNS context are shown in Tables 5.5 and 5.6. Instances are classified according to their size, expressed by the number of requests ($|R|$), available vehicles ($|V|$) and health centers ($|H|$). In addition, synthetic instances are sub-categorized by difficulty which is related to the number of constraints and the availability of vehicles. The column **BKO** indicates the best known optimum for each instance. For each approach, the best result of all the runs is reported in the column Sol. The "$\star$" symbol indicates that the solution has been proven optimal in at least one run. The column **rdist** indicates the average relative distance between each run at the end of the search. Figure 5.11 shows the evolution of the average relative distance for all instances, for each run during the search.

The results show that both implementations of the sequence variable are able to successfully outperform the dedicated SCHED+ODS approach on the PTP. The prefix sequence variable approach (PSV) manages to obtain good results on the easier instances but is outperformed by both the CP Optimizer approach (LIU_CPO) and the insertion sequence approaches (ISEQ) on larger or harder instances. In terms of general performances, it is on par with the

Table 5.5: Comparison of performances on the synthetic instances of the PTP with LNS ("Sol" indicates the best solution, "rdist" corresponds to the average relative distance at the end of the search)

Instance						LIU_CPO		SCHED+ODS		PSEQ		ISEQ+FF		ISEQ+SDS	
Difficulty	Name	$\|H\|$	$\|V\|$	$\|P\|$	BKO	Sol	rdist	Sol	rdist	Sol	rdist	Sol	rdist	Sol	rdist
easy	RAND-E-1	4	2	16	⋆15	⋆15	0	⋆15	0	⋆15	0	⋆15	0	⋆15	0
easy	RAND-E-2	8	4	32	⋆32	⋆32	0	⋆32	0	⋆32	0	⋆32	0	⋆32	0
easy	RAND-E-3	12	5	48	⋆28	⋆28	0	⋆28	0	⋆28	0	⋆28	0	⋆28	0
easy	RAND-E-4	16	6	64	⋆64	⋆64	0	62	0.042	⋆64	0	⋆64	0	⋆64	0
easy	RAND-E-5	20	8	80	⋆80	79	0.013	75	0.075	⋆80	0.011	⋆80	0.001	⋆80	0
easy	RAND-E-6	24	9	96	⋆96	⋆96	0	94	0.028	⋆96	0	⋆96	0	⋆96	0
easy	RAND-E-7	28	10	112	⋆112	⋆112	0	106	0.064	⋆112	0.009	⋆112	0.001	⋆112	0
easy	RAND-E-8	32	12	128	⋆128	⋆128	0	⋆128	0	⋆128	0	⋆128	0	⋆128	0
easy	RAND-E-9	36	14	144	⋆144	⋆144	0	142	0.014	⋆144	0	⋆144	0	⋆144	0
easy	RAND-E-10	40	16	160	⋆160	159	0.006	157	0.019	⋆160	0	⋆160	0	⋆160	0
medium	RAND-M-1	8	2	16	⋆12	⋆12	0	11	0.083	⋆12	0	⋆12	0	⋆12	0
medium	RAND-M-2	16	3	32	⋆20	⋆20	0	⋆20	0.05	⋆20	0.015	⋆20	0	⋆20	0
medium	RAND-M-3	24	4	48	⋆35	⋆35	0	33	0.091	⋆35	0.017	⋆35	0.009	⋆35	0.014
medium	RAND-M-4	32	4	64	⋆42	41	0.024	39	0.152	40	0.069	⋆42	0.021	⋆42	0.01
medium	RAND-M-5	40	5	80	⋆69	⋆69	0	59	0.181	66	0.071	67	0.049	67	0.051
medium	RAND-M-6	48	5	96	⋆61	60	0.016	51	0.184	40	0.344	⋆61	0.038	⋆61	0.028
medium	RAND-M-7	56	6	112	⋆77	75	0.029	62	0.213	46	0.403	⋆77	0.036	75	0.042
medium	RAND-M-8	64	8	128	⋆96	⋆96	0	84	0.147	72	0.25	95	0.032	95	0.025
medium	RAND-M-9	72	8	144	⋆99	94	0.054	82	0.199	65	0.343	98	0.027	⋆99	0.023
medium	RAND-M-10	80	9	160	⋆117	112	0.044	100	0.162	75	0.359	⋆117	0.016	⋆117	0.019
hard	RAND-H-1	16	2	16	⋆8	⋆8	0	⋆8	0	⋆8	0	⋆8	0	7	0.125
hard	RAND-H-2	32	3	32	⋆19	⋆19	0	⋆19	0.042	⋆19	0	⋆19	0	⋆19	0
hard	RAND-H-3	48	4	48	⋆35	34	0.029	32	0.117	⋆35	0.026	34	0.031	34	0.04
hard	RAND-H-4	64	4	64	⋆25	⋆25	0	24	0.128	⋆25	0	24	0.056	24	0.06
hard	RAND-H-5	80	5	80	⋆48	⋆48	0	45	0.121	⋆48	0.194	47	0.04	⋆48	0.023
hard	RAND-H-6	96	5	96	⋆47	⋆47	0	41	0.194	⋆47	0.147	45	0.06	45	0.066
hard	RAND-H-7	112	6	112	⋆44	41	0.068	40	0.148	43	0.318	⋆44	0.034	⋆44	0.009
hard	RAND-H-8	128	8	128	⋆89	86	0.034	78	0.151	58	0.348	87	0.038	⋆89	0.034
hard	RAND-H-9	144	8	144	⋆85	83	0.024	75	0.141	54	0.365	84	0.04	84	0.029
hard	RAND-H-10	160	8	160	⋆83	79	0.048	73	0.143	81	0.073	82	0.024	⋆83	0.022

scheduling model.

As can be observed, the approach using the insertion sequence variable obtain slightly better result than the approach using the state-of-the-art CP Optimizer. Note that the comparison with CP Optimizer is not straightforward as it is mostly black box and its interface does not offer much control over its behavior. However, despite the adaptive LNS search [LG07] and the advanced techniques (failure directed search [VLS15], objective landscapes [Lab18b]) used by CP Optimizer, the sequence variable approach is competitive in an LNS setting.

The difference of searches used by the insertion sequence variable (First Fail (FF) or Slack Driven (SDS)) have a low impact on the performances. The SDS seems a bit better. An interesting result is that both searches are especially good on the real world instances and SDS even manages to prove the optimum on all real instances.

The general behavior of the different methods during the search can be observed on Figure 5.11. We can see that both the scheduling model and the prefix sequence variable model start to stagnate at around 20% of the relative distance and are unable to go much further. The CP optimizer approach dom-

Table 5.6: Comparison of performances on the synthetic instances of the PTP with LNS ("Sol" indicates the best solution, "rdist" corresponds to the average relative distance at the end of the search)

Instance					LIU_CPO		SCHED+ODS		PSEQ		ISEQ+FF		ISEQ+SDS	
Name	$\|H\|$	$\|V\|$	$\|P\|$	BKO	best	rdist	best	rdist	best	rdist	best	rdist	best	rdist
REAL-1	1	9	2	⋆2	⋆2	0	⋆2	0	⋆2	0	⋆2	0	⋆2	0
REAL-2	1	9	2	⋆2	⋆2	0	⋆2	0	⋆2	0	⋆2	0	⋆2	0
REAL-3	3	9	3	⋆1	⋆1	0	⋆1	0	⋆1	0	⋆1	0	⋆1	0
REAL-4	2	9	4	⋆4	⋆4	0	⋆4	0	⋆4	0	⋆4	0	⋆4	0
REAL-5	5	9	21	⋆21	⋆21	0	⋆21	0	⋆21	0	⋆21	0	⋆21	0
REAL-6	5	9	22	⋆22	⋆22	0	⋆22	0	⋆22	0	⋆22	0	⋆22	0
REAL-7	5	9	23	⋆23	⋆23	0	⋆23	0	⋆23	0	⋆23	0	⋆23	0
REAL-8	7	9	24	⋆24	⋆24	0	⋆24	0	⋆24	0	⋆24	0	⋆24	0
REAL-9	15	9	45	⋆44	⋆44	0	⋆44	0	⋆44	0	⋆44	0	⋆44	0
REAL-10	26	9	99	⋆98	⋆98	0	⋆98	0	⋆98	0	⋆98	0	⋆98	0
REAL-11	22	9	100	⋆92	⋆92	0	91	0.016	⋆92	0.001	⋆92	0	⋆92	0
REAL-12	32	9	101	⋆100	⋆100	0	98	0.02	⋆100	0.008	⋆100	0	⋆100	0
REAL-13	37	9	110	⋆107	⋆107	0	104	0.038	⋆107	0.006	⋆107	0	⋆107	0
REAL-14	28	9	111	⋆102	⋆102	0	⋆102	0.011	⋆102	0	⋆102	0	⋆102	0
REAL-15	35	9	122	⋆119	118	0.008	109	0.097	115	0.04	118	0.008	118	0.01
REAL-16	36	9	123	⋆117	⋆117	0	111	0.068	107	0.085	⋆117	0	⋆117	0
REAL-17	42	9	128	⋆122	⋆122	0	114	0.078	120	0.026	⋆122	0	⋆122	0
REAL-18	31	9	130	⋆126	⋆126	0	121	0.053	125	0.015	⋆126	0	⋆126	0
REAL-19	34	9	131	⋆122	121	0.008	116	0.084	121	0.017	⋆122	0.005	⋆122	0.001
REAL-20	34	9	134	⋆129	128	0.014	117	0.117	125	0.047	⋆129	0.01	⋆129	0.005
REAL-21	39	9	136	⋆127	126	0.008	120	0.075	125	0.023	⋆127	0.001	⋆127	0
REAL-22	31	9	138	⋆131	⋆131	0	123	0.087	128	0.031	⋆131	0.002	⋆131	0
REAL-23	31	9	139	⋆136	134	0.015	123	0.121	131	0.048	⋆136	0.012	⋆136	0.007
REAL-24	37	9	139	⋆124	120	0.035	112	0.115	120	0.043	123	0.017	⋆124	0.006
REAL-25	39	9	139	⋆135	131	0.03	128	0.076	119	0.119	⋆135	0.003	⋆135	0.002
REAL-26	38	9	140	⋆131	⋆131	0	120	0.102	110	0.16	⋆131	0.002	⋆131	0.001
REAL-27	35	9	147	⋆139	138	0.007	129	0.084	121	0.129	138	0.009	⋆139	0.006
REAL-28	34	9	151	⋆145	144	0.007	129	0.119	117	0.193	⋆145	0.007	⋆145	0.006
REAL-29	39	9	155	⋆147	144	0.02	129	0.139	120	0.184	146	0.012	⋆147	0.009
REAL-30	41	9	159	⋆149	138	0.079	134	0.127	116	0.221	148	0.014	⋆149	0.008

inates during the first 10 seconds of the search but is quickly outperformed by both the insertion sequence based methods as its slope decreases. An interesting observation is that after some time LIU_CPO descends again faster to reach the same relative distance than the insertion based models. Our hypothesis is that it is thanks to the failure directed search [VLS15] that is used by CP optimizer when the LNS starts to stagnate. It shows a possible direction to improve the search for the insertion sequence variables.

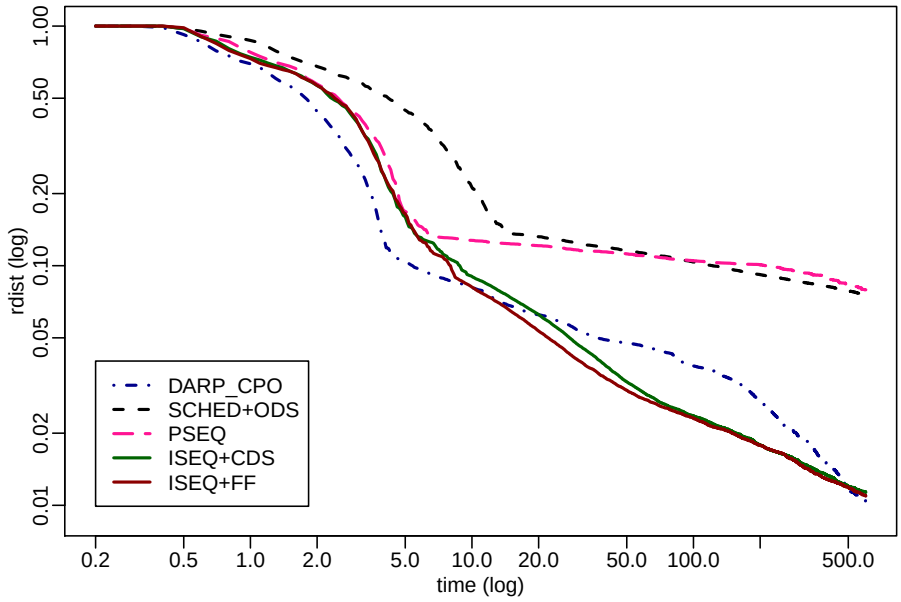

Figure 5.11: Average relative distance in function of time on the PTP with LNS

5.7.2 PTP with DFS

The next experiment compares the approaches in a DFS setting. It allows us to compare the models with a similar search and have a fairer comparison with CP Optimizer which uses a standard first fail heuristic in this context. The results are shown in Tables 5.7 and 5.8. Figure 5.12 shows the anytime behavior.

Table 5.7: Comparison of performances on the synthetic instances of the PTP with DFS ("Sol" indicates the best solution, "rdist" corresponds to the average relative distance at the end of the search)

Instance						LIU_CPO		SCHED+ODS		PSEQ		ISEQ+FF		ISEQ+SDS	
Difficulty	Name	$\|H\|$	$\|V\|$	$\|P\|$	BKO	Sol	rdist	Sol	rdist	Sol	rdist	Sol	rdist	Sol	rdist
easy	RAND-E-1	4	2	16	⋆15	⋆15	0	14	0.067	⋆15	0	⋆15	0	⋆15	0
easy	RAND-E-2	8	4	32	⋆32	18	0.438	30	0.063	⋆32	0	⋆32	0	⋆32	0.003
easy	RAND-E-3	12	5	48	⋆28	22	0.214	26	0.071	0	1	⋆28	0	⋆28	0.004
easy	RAND-E-4	16	6	64	⋆64	25	0.609	53	0.172	⋆64	0	⋆64	0.023	61	0.092
easy	RAND-E-5	20	8	80	⋆80	18	0.775	66	0.175	77	0.038	⋆80	0.048	73	0.114
easy	RAND-E-6	24	9	96	⋆96	23	0.76	87	0.094	⋆96	0	⋆96	0	94	0.047
easy	RAND-E-7	28	10	112	⋆112	22	0.804	90	0.196	0	1	111	0.051	103	0.1
easy	RAND-E-8	32	12	128	⋆128	26	0.797	120	0.063	0	1	⋆128	0	⋆128	0.012
easy	RAND-E-9	36	14	144	⋆144	21	0.854	129	0.104	⋆144	0	⋆144	0.029	139	0.066
easy	RAND-E-10	40	16	160	⋆160	21	0.869	135	0.156	0	1	⋆160	0.011	149	0.079
medium	RAND-M-1	8	2	16	⋆12	⋆12	0	9	0.25	⋆12	0	⋆12	0	⋆12	0
medium	RAND-M-2	16	3	32	⋆20	⋆20	0	15	0.25	19	0.05	⋆20	0	⋆20	0
medium	RAND-M-3	24	4	48	⋆35	22	0.371	29	0.171	33	0.057	33	0.223	34	0.094
medium	RAND-M-4	32	4	64	⋆42	23	0.452	31	0.262	37	0.119	34	0.259	38	0.157
medium	RAND-M-5	40	5	80	⋆69	19	0.725	46	0.333	64	0.072	59	0.196	57	0.197
medium	RAND-M-6	48	5	96	⋆61	11	0.82	45	0.262	0	1	55	0.152	54	0.192
medium	RAND-M-7	56	6	112	⋆77	19	0.753	49	0.364	68	0.117	65	0.219	64	0.196
medium	RAND-M-8	64	8	128	⋆96	16	0.833	68	0.292	0	1	80	0.207	83	0.18
medium	RAND-M-9	72	8	144	⋆99	25	0.747	63	0.364	0	1	76	0.276	86	0.216
medium	RAND-M-10	80	9	160	⋆117	14	0.88	83	0.291	0	1	94	0.226	100	0.17
hard	RAND-H-1	16	2	16	⋆8	⋆8	0	6	0.25	⋆8	0	5	0.375	7	0.275
hard	RAND-H-2	32	3	32	⋆19	⋆19	0	16	0.158	⋆19	0	17	0.105	18	0.068
hard	RAND-H-3	48	4	48	⋆35	21	0.4	25	0.286	33	0.057	26	0.297	30	0.174
hard	RAND-H-4	64	4	64	⋆25	22	0.12	21	0.16	⋆25	0	23	0.136	24	0.1
hard	RAND-H-5	80	5	80	⋆48	21	0.563	34	0.292	0	1	35	0.323	41	0.179
hard	RAND-H-6	96	5	96	⋆47	21	0.553	32	0.319	0	1	38	0.274	37	0.26
hard	RAND-H-7	112	6	112	⋆44	18	0.591	34	0.227	0	1	31	0.318	40	0.198
hard	RAND-H-8	128	8	128	⋆89	29	0.674	65	0.27	0	1	56	0.393	74	0.242
hard	RAND-H-9	144	8	144	⋆85	17	0.8	60	0.294	0	1	62	0.312	71	0.226
hard	RAND-H-10	160	8	160	⋆83	17	0.795	62	0.253	0	1	58	0.373	66	0.223

The experiment in the DFS setting, where the advanced search of CP Optimizer is not used, suggests that the difference is mainly due to the modeling and propagation. Indeed, even our generic search outperforms CP Optimizer in this setting.

Another striking result is that the prefix sequence model is particularly bad in this setting, having difficulties even finding an initial solution on most instances. This is somewhat expected as the sequence selection search has mainly been designed to be used in a LNS. Its heuristic considers the stops independently, without taking into account the requests. This leads to decisions early in the search tree which impact other stops that become required but are considered far further in the search tree. This results show the shortcom-

Table 5.8: Comparison of performances on the real instances of the PTP with DFS ("Sol" indicates the best solution, "rdist" corresponds to the average relative distance at the end of the search)

Instance					LIU_CPO		SCHED+ODS		PSEQ		ISEQ+FF		ISEQ+SDS	
Name	$\|H\|$	$\|V\|$	$\|P\|$	BKO	Sol	rdist	Sol	rdist	Sol	rdist	Sol	rdist	Sol	rdist
REAL-1	1	9	2	⋆2	⋆2	0	⋆2	0	⋆2	0	⋆2	0	⋆2	0
REAL-2	1	9	2	⋆2	⋆2	0	⋆2	0	⋆2	0	⋆2	0	⋆2	0
REAL-3	3	9	3	⋆1	⋆1	0	⋆1	0	⋆1	0	⋆1	0	⋆1	0
REAL-4	2	9	4	⋆4	⋆4	0	⋆4	0	⋆4	0	⋆4	0	⋆4	0
REAL-5	5	9	21	⋆21	⋆21	0	⋆21	0	⋆21	0	⋆21	0	⋆21	0
REAL-6	5	9	22	⋆22	12	0.455	⋆22	0	⋆22	0	⋆22	0	⋆22	0
REAL-7	5	9	23	⋆23	16	0.304	⋆23	0	⋆23	0	⋆23	0	⋆23	0
REAL-8	7	9	24	⋆24	13	0.458	⋆24	0	⋆24	0	⋆24	0	⋆24	0
REAL-9	15	9	45	⋆44	14	0.682	⋆44	0	⋆44	0	⋆44	0	⋆44	0
REAL-10	26	9	99	⋆98	16	0.837	97	0.01	0	1	⋆98	0	⋆98	0.003
REAL-11	22	9	100	⋆92	26	0.717	83	0.098	90	0.022	⋆92	0.021	90	0.036
REAL-12	32	9	101	⋆100	23	0.77	94	0.06	96	0.04	⋆100	0	99	0.029
REAL-13	37	9	110	⋆107	15	0.86	95	0.112	0	1	106	0.057	103	0.059
REAL-14	28	9	111	⋆102	26	0.745	95	0.069	0	1	⋆102	0	101	0.047
REAL-15	35	9	122	⋆119	14	0.882	95	0.202	0	1	109	0.126	110	0.105
REAL-16	36	9	123	⋆117	14	0.88	97	0.171	0	1	⋆117	0.083	111	0.079
REAL-17	42	9	128	⋆122	14	0.885	97	0.205	0	1	111	0.146	112	0.098
REAL-18	31	9	130	⋆126	13	0.897	104	0.175	0	1	123	0.069	119	0.075
REAL-19	34	9	131	⋆122	10	0.918	103	0.156	0	1	117	0.089	116	0.095
REAL-20	34	9	134	⋆129	16	0.876	103	0.202	0	1	117	0.125	115	0.122
REAL-21	39	9	136	⋆127	10	0.921	106	0.165	0	1	120	0.121	116	0.107
REAL-22	31	9	138	⋆131	33	0.748	101	0.229	0	1	120	0.117	123	0.095
REAL-23	31	9	139	⋆136	12	0.915	100	0.265	0	1	119	0.152	121	0.139
REAL-24	37	9	139	⋆124	11	0.911	90	0.274	0	1	112	0.144	112	0.136
REAL-25	39	9	139	⋆135	13	0.904	111	0.178	0	1	128	0.114	126	0.101
REAL-26	38	9	140	⋆131	16	0.878	101	0.229	0	1	119	0.136	121	0.116
REAL-27	35	9	147	⋆139	24	0.827	106	0.237	0	1	120	0.177	125	0.13
REAL-28	34	9	151	⋆145	13	0.91	108	0.255	0	1	137	0.143	127	0.148
REAL-29	39	9	155	⋆147	10	0.932	109	0.259	0	1	124	0.186	131	0.148
REAL-30	41	9	159	⋆149	11	0.926	106	0.289	0	1	126	0.179	130	0.15

ings of the prefix based approach and the advantages of the insertion based variable that allows much more flexibility in the search. An interesting observation is that despite its shortcomings, the prefix based approach is able to obtain good results and even outperform other methods on a few specific instances. This might be due to the greedy nature of its heuristic that works well in some specific configurations, for example if the stops of a same request are close to each other in time.

The anytime behavior of the methods (Figure 5.12 shows that the insertion sequence model clearly outperforms the other models given enough search time. The scheduling model is faster at the beginning of the search but starts to stagnate quickly. The difference between the first fail and slack driven heuristics is more important in this DFS context, the later one being able to get closer to the optimum.

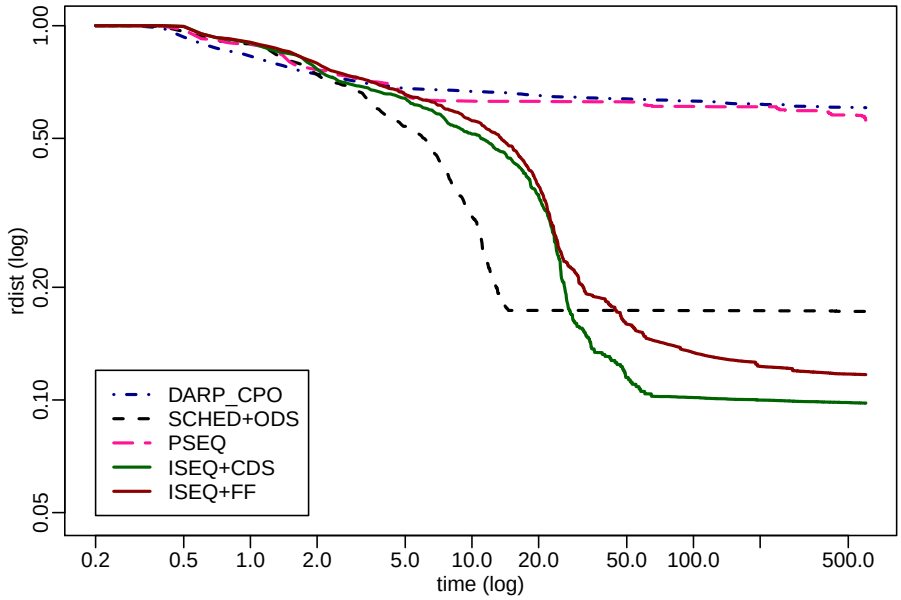

Figure 5.12: Average relative distance in function of time on the PTP with DFS

5.7.3 DARP

For the DARP, the insertion sequence approach was compared with:

1. the *LNS with First Feasible Probabilistic Acceptance*(LNS-FFPA) model and heuristic proposed in [JV11];

2. an implementation of our model with the sequence variables and interval variables of *CP Optimizer* which is referred as DARP_CPO.

Note that due to the difficulties of the prefix sequence variable to find an initial solution, it was not considered on this problem.

The approaches were run on 68 DARP instances from [CL03a; Cor06] that are available at [Bra22]. The results are available in Tables 5.9 and 5.10. As for the two precedent experiments, the average relative distance during the search is used to show the general behavior of the different approaches in Figure 5.13.

These results show that the three approaches using sequence variables are not able to compete with the LNS-FFPA approach. This method uses a search technique heavily tailored to this specific version of the DARP and even enforces some constraints outside of the model, during the search. This allows the LNS-FFPA to be highly efficient but makes it hard to easily adapt

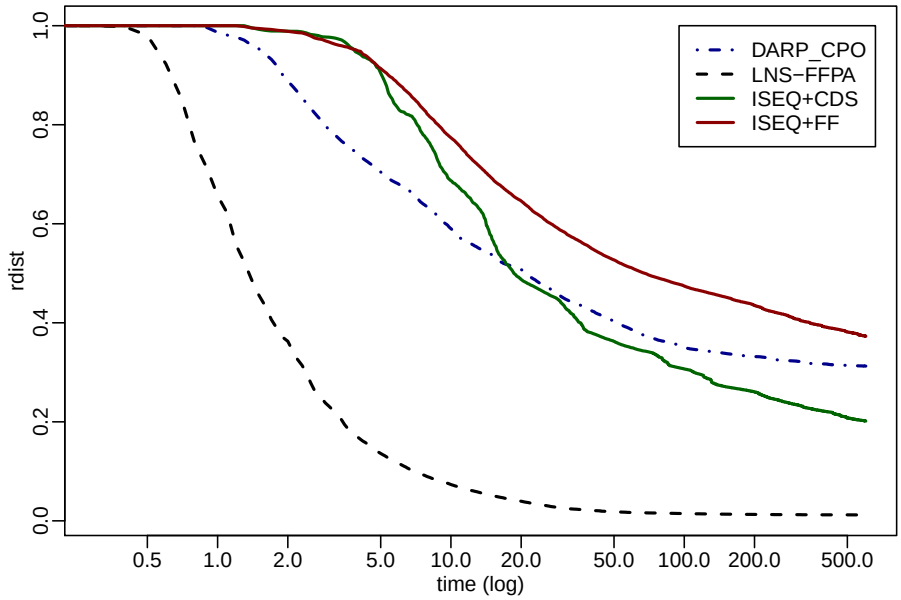

Figure 5.13: Average relative distance in function of time on the DARP

to other problems. On the other hand, the other approaches are more generic and their components can easily be reused for other problems.

The comparison of the two insertion sequence approaches with the CP optimizer approach shows that the cost driven heuristic (CDS) gives an edge over CP Optimizer (DARP_CPO) at the end of the search while the general first fail heuristic (FF) is the least efficient.

Table 5.9: Comparison of performances on the DARP (1/2, "Sol" indicates the best solution, "rdist" corresponds to the average relative distance at the end of the search)

Instance	BKO	DARP_CPO		ISEQ+FF		ISEQ+CDS		LNS-FFPA	
		Sol	rdist	Sol	rdist	Sol	rdist	Sol	rdist
a2-16	294	294	0	294	0	294	0.005	294	0
a2-20	344	345	0.008	345	0.015	348	0.032	344	0
a2-24	431	431	0	431	0.013	431	0.016	431	0
a3-18	300	301	0.009	300	0.006	300	0.006	300	0
a3-24	190	200	0.068	190	0.017	193	0.039	195	0.039
a3-30	494	504	0.066	495	0.025	496	0.013	494	0.004
a3-36	583	∅	1	583	0.025	583	0.028	583	0
a4-16	282	283	0.01	283	0.01	283	0.01	282	0
a4-24	375	375	0	375	0.005	375	0.009	375	0
a4-32	485	487	0.009	485	0.026	485	0.023	485	0
a4-36	291	∅	1	293	0.053	293	0.045	291	0.005
a4-40	557	566	0.038	571	0.063	564	0.057	557	0.002
a4-48	670	691	0.064	691	0.098	678	0.046	670	0.01
a5-40	499	537	0.11	509	0.056	504	0.031	499	0.003
a5-48	307	∅	1	311	0.038	307	0.022	308	0.014
a5-50	689	690	0.003	705	0.079	691	0.046	690	0.008
a5-60	809	844	0.081	871	0.202	830	0.071	809	0.012
a6-48	604	636	0.082	616	0.1	613	0.056	604	0.003
a6-60	830	910	0.163	920	0.272	842	0.046	830	0.009
a6-72	522	∅	1	695	0.4	523	0.026	522	0.016
a7-56	827	858	0.074	857	0.117	853	0.085	827	0.009
a7-70	923	978	0.109	1006	0.241	945	0.078	923	0.01
a7-72	550	∅	1	762	0.407	579	0.08	550	0.011
a7-84	1218	1309	0.152	1453	0.447	1278	0.157	1218	0.023
a8-108	712	∅	1	∅	1	∅	1	712	0.081
a8-64	758	810	0.096	866	0.248	769	0.046	758	0.006
a8-80	959	1018	0.099	1221	0.486	1004	0.099	959	0.015
a8-96	1255	1346	0.114	1667	0.541	1321	0.103	1255	0.012
a9-96	613	∅	1	995	0.587	677	0.123	613	0.014
a10-144	929	∅	1	∅	1	1075	0.213	929	0.021
a11-120	690	∅	1	1153	0.629	792	0.227	690	0.011
a13-144	868	∅	1	1582	0.691	1091	0.255	868	0.01

Table 5.10: Comparison of performances on the DARP (2/2, "Sol" indicates the best solution, "rdist" corresponds to the average relative distance at the end of the search)

Instance	BKO	DARP_CPO		ISEQ+FF		ISEQ+CDS		LNS-FFPA	
		Sol	rdist	Sol	rdist	Sol	rdist	Sol	rdist
b2-16	309	310	0.021	∅	1	∅	1	309	0
b2-20	332	333	0.031	∅	1	∅	1	332	0
b2-24	444	445	0.014	∅	1	∅	1	444	0.003
b3-18	301	302	0.02	∅	1	∅	1	301	0.004
b3-24	164	200	0.202	164	0.042	167	0.035	169	0.034
b3-30	531	537	0.071	∅	1	∅	1	531	0.01
b3-36	603	609	0.029	609	0.057	605	0.029	603	0
b4-16	296	298	0.02	297	0.01	297	0.01	296	0
b4-24	371	371	0	∅	1	∅	1	371	0
b4-32	494	509	0.072	495	0.031	501	0.062	494	0
b4-36	248	311	0.22	259	0.07	254	0.055	253	0.032
b4-40	656	662	0.032	∅	1	∅	1	656	0.01
b4-48	675	710	0.118	690	0.2	678	0.035	675	0.002
b5-40	614	657	0.135	616	0.035	618	0.032	614	0.003
b5-48	299	349	0.163	325	0.164	299	0.052	313	0.059
b5-50	765	781	0.048	∅	1	775	0.059	765	0.008
b5-60	909	975	0.14	933	0.126	929	0.073	909	0.008
b6-48	716	762	0.185	∅	1	722	0.059	716	0.012
b6-60	861	923	0.145	920	0.179	895	0.134	861	0.019
b6-72	497	∅	1	716	0.418	514	0.07	497	0.016
b7-56	829	858	0.07	865	0.135	850	0.076	829	0.006
b7-70	926	978	0.107	1028	0.255	951	0.076	926	0.006
b7-72	520	∅	1	763	0.413	531	0.055	520	0.011
b7-84	1220	1309	0.132	1446	0.385	1281	0.153	1220	0.019
b8-108	654	∅	1	1106	0.556	705	0.106	654	0.014
b8-64	853	904	0.107	931	0.207	885	0.104	853	0.01
b8-80	1052	1175	0.208	1255	0.372	1106	0.127	1052	0.009
b8-96	1211	1318	0.132	1517	0.422	1319	0.15	1211	0.018
b9-96	598	757	0.222	968	0.543	651	0.107	598	0.015
b10-144	908	∅	1	∅	1	1129	0.587	908	0.065
b11-120	674	∅	1	1165	0.639	813	0.205	674	0.01
b13-144	848	∅	1	1590	0.7	1100	0.36	848	0.015

5.7.4 Constraint parameters

Several values were tested for the parameters of the `TransitionTimes` constraint by using the methodology proposed in [VLS18]. It consists in storing the search tree obtained with the weakest filtering and replaying it with the constraints and parameters to test. The impact of the `Cumulative` constraint was also tested by comparing it to a simple checker.

Table 5.11 presents the results of this experiment on 3 medium sized PTP instances in a DFS setting. The instances are expressed in terms of the number of hospitals (h), number of available vehicles (v) and number of patients (p). The values are displayed in terms of percentage compared to the base case (the parameter value for the first column). The first row corresponds to the percentage of size (in terms of the number of nodes) of the new search tree compared to the base case. The second row consists in the percentage of time taken to explore the new search tree. For example, on the Hard instance, for a depth d of 2, the search tree is 76.26% smaller which results in an exploration 63.67% faster. Each parameter was tested independently with the others set to their default values.

Table 5.11: Number of nodes explored (top) and time taken (bottom) with various parameter values

Instance				ρ				d					cache		Cumul.	
Set	h	v	p	0	10	20	∞	1	2	3	6	∞	×	√	×	√
Easy	24	9	96	100	100	100	100	100	100	100	100	100	100	100	100	100
				100	96.27	92.27	101	100	102.53	96.68	93.87	91.73	100	95.39	100	126.61
Medium	48	5	96	100	100	100	100	100	0.01	0.01	0.01	0.01	100	100	100	0.01
				100	80.79	55.67	53.07	100	0.05	0.06	0.05	0.06	100	77.24	100	0.01
Hard	96	5	96	100	100	100	55.35	100	76.26	70.31	64.48	59.86	100	100	100	0.01
				100	85.06	56.47	22.74	100	63.67	53.8	38.8	51.93	100	91.44	100	0.03

As can be observed, the constraints have an important impact on both the size of the search tree and the search time for the medium and difficult instances. The easy instance search tree was not affected by the constraints. Note that an increase in depth may result in a faster search despite having the same search tree size (such as for the Easy instance). This is most likely due to the cache that is filled faster in the first calls to Algorithm 1 and thus allows smaller searches in the subsequent calls which results in a gain of time over the whole propagation.

5.8 conclusion

In this chapter, the usage of sequence variables on routing and scheduling problems was discussed. Two possible implementations for a sequence a variable were proposed. The first one uses a growing prefix to represent the internal sequence. The second one is based on a linked representation that allows insertion of new elements at any place in the growing sequence. In

addition, several constraints that use these variables were proposed and their filtering algorithms detailed. Finally, these variables and constraints were used to model and solve the patient transportation and dial-a-ride problems. Their performances were analyzed and compared with other approaches on datasets of PTP and DARP instances.

The contributions in this chapter are the following:

- The definition and implementation of the Prefix Sequence Variable (PSV) (Section 5.3).
- The definition and implementation of the Insertion Sequence Variable (ISV) (Section 5.4).
- A set of dedicated constraints for the sequence variables as well as filtering algorithms that leverage the information maintained by the variables (Section 5.5)
- Models for the PTP and DARP that use the sequence variables and their constraints (5.6)
- An experimental evaluation of these models as well as a comparison with other approaches for the PTP and DARP.

The experimental results show that the insertion sequence variable is competitive with other state-of-the-art approaches and outperforms them on the PTP. The insertion-based approach is even able to outperform the adapted model for the PTP from [LAB18] that was the most successful of the tested approaches in chapter 4. The experiments on several instances with various parameters for the *TransitionTimes* constraint shows that the filtering algorithm proposed is able to improve the size of the search tree and the search time. These results highlight the efficiency of the sequence variables on hybrid routing and scheduling problems such as the PTP and DARP and point to many opportunities to successfully use and improve the sequence variables and constraints on combinatorial problems.

Conclusion 6

The work done in this thesis was focused on the efficient resolution of routing and scheduling problems through Constraint Programming. Two main research directions were explored. First, a generic adaptive search, destined to be used in a black-box context. Second, the use of dedicated modeling and search techniques to solve specific problems.

Adaptive Large Neighborhood Search

The first research direction explored was the use of generic search techniques to solve CP problems in a black-box context. In particular, an adaptive variant of the Large Neighborhood Search [LG07] was used with a portfolio of destroy and repair heuristics. This approach consists in iteratively applying a partial relaxation followed by a search to the current best solution of the problem in order to explore different parts of the search space. A relaxation and a search heuristic is selected from the portfolio at each iteration. These operators have their selection biased during the search based on their performances.

The implementation proposed uses an adapted selection mechanism designed to better deal with the different running times of the operators. It was tested on a set of instances from 10 different problems with operators tailored for specific cases. The results show an improvement of the performances compared with the original implementation of the ALNS. Furthermore, our implementation is able to select appropriate relaxation and search heuristics tailored for the problems encountered. This validates the use of such a technique in a black box context.

Patient Transportation Problem

The Patient Transportation Problem was formalized based on a case study studied as part of the PRESupply project. This problem, which is a variant of the DARP, consists in planning the operations of a fleet of vehicles in order to transport patients to and from medical appointments. Several resolution methods were explored to solve this problem. A scheduling model was proposed and evaluated against the classical successor approach generally used to solve routing problems. It was demonstrated more efficient on the PTP

which combines both routing and scheduling aspects. Furthermore, a dedicated search technique called Optional Decision Search (ODS) was developed and seems to have contributed to the success of the scheduling model. However, comparisons with another model published independently and implemented in the CP optimizer solver which was proposed to solve a variant of the PTP [LAB18] showed that our approach is outperformed. This led us to the hypothesis that the powerful scheduling oriented modeling features of CP Optimizer allows a better modeling of hybrid problems such as the PTP which leads to a gain in term of performances.

Sequence Variables

Following the results of the PTP models, we investigated the use of dedicated modeling features for such problems. The result was the development of a generic sequence variable that represents a set of elements to order. While similar variables are already part of several commercial solvers including CP Optmizer, their implementation has not been published. Furthermore, the sequence variable of CP Optimizer requires activity variables, making it difficult to use on other problems than scheduling.

We proposed two different implementations for our sequence variable. Both are generic and easy to implement. They are based on an extension of the set variable with an internal growing sequence. The prefix sequence variable works by appending elements to an ordered prefix. The insertion sequence variable uses a linked structure that allows the insertion of elements in any part of the internal sequence. Additionally, possibles insertions are maintained during the search which allows propagation algorithms to use this information. We also developed dedicated constraints that allow to model problems such as the PTP.

We tested our implementations on the DARP and PTP problems against state-of-the-art dedicated approaches and the model of Liu et al. [LAB18]. The results show the competitiveness of our approach. While not able to obtain results as good as the state of the art LNS-FFPA approach on the DARP [JV11], our sequence variable model is able to compete with other generic approaches. On the PTP, it is currently the most efficient approach, being able to outperform Liu et al. CP Optimizer model. The insertion sequence model is now in use at the CSD to plan their transportation services.

6.1 Further Work

Many research perspectives are open on the topics explored in this thesis. Some of them have already been or are currently being investigated.

Adaptive Large Neighborhood Search

A meta-analysis of several publications on the topic of ALNS has been performed in [TSH21]. This works highlights the results provided by our approach and confirms the efficiency of ALNS method in a black-box context. There are many research opportunities open in this domain.

In particular, the portfolio of operators used could be expanded with new relaxation and search techniques. The inclusion of problem focused heuristics among the portfolio is promising. Furthermore, additional research could be done on the adaptive mechanism that selects the operators during the search. The ALNS approach has so far been used to solve COPs but recent work in [Li+22] shows that a portfolio based approach can be successfully applied to CSPs in a black-box context. Finally, the ALNS approach could also be hybridized with other learning methods to pre-select a subset of operators or parameters before the search based on the problem features or even help the selection during the search [KGM12; DAA21].

Patient Transportation Problem

While the model based on insertion sequence variables solves efficiently the PTP, improvements are certainly possible. A dedicated approach such as the LNS-FFPA [JV11] could be adapted to this particular problem. Several variants of the problem are possible and could also be studied. In particular, multiple objectives could be considered at the same time. In order to solve such cases, the approaches proposed would have to be adapted to support multi-objective optimization.

In addition, while the problem studied assumes fixed requests known beforehand, real-life application often impose dynamic aspects. Indeed, requests may be issued during the day of operations rather than beforehand. Some requests could also be modified or deleted during the operations. In [PCP20], the authors explore recovery strategies for dealing with such modifications during the operations given an initial planning. Another way to deal with this dynamic aspect is to directly integrate it in the CP model by using some kind of destroy and repair procedure when confronted to changes to the initial requests. Additionally, other constraints or objectives could be introduced to make the solution more robust or flexible to changes.

Finally, the optional decision search strategy could certainly be used for other applications such as packing problems [SO08; SO11; HK13].

Sequence Variables

On the topic of sequence variables, work has already been done in [DSV22] to improve the performances of the insertion sequence variable. The implemen-

tation proposed used a simplified representation of the set domain as well as new filtering algorithms. It was able to find new optimums for several instances of the TSPTW from well known sets.

Currently, several research directions are explored on the sequence variables. One of them is the addition of potential successors to the domain of the variable. This would allow to build a precedence graph and use it in filtering algorithms for precedence and time-related constraints.

The filtering algorithms for some of the constraints proposed could also be improved. For example, a lighter filtering based on some form of path relaxation could be considered for the `TransitionTimes` constraint. Besides that, other search heuristics could be elaborated in order to better deal with symmetries in the search.

Additionally, the use of sequence variables is investigated for solving pure scheduling problems such as the Job Shop Problem [YN97]. In particular, several propagation algorithms from the scheduling domain could be adapted to the sequence variable [MDD07; MV08; SW10; Vil11].

As the sequence variable is generic by design, it could be used for other problems. Many combinatorial problems consist in ordering elements in an optimal arrangement [Fle90; GRW08; Rig20] or could benefit from a sequential representation [PHW07; NZ14; Fou+17]. With dedicated constraints, sequence variables would allow powerful modeling in CP to solve such problems.

Bibliography

[AAL03] E. Aarts, E. H. Aarts, and J. K. Lenstra. *Local Search in Combinatorial Optimization*. Princeton University Press, 2003.

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[AB93] A. Aggoun and N. Beldiceanu. "Extending CHIP in Order to Solve Complex Scheduling and Placement Problems". In: *Mathematical and computer modelling* 17.7 (1993), pp. 57–73.

[Abd82] L. Abdelkader. "Scheduling-the Notions of Hump, Compulsory Parts and Their Use in Cumulative Problems". In: *Comptes rendus de l'Academie des sciences*. I 294.6 (1982), pp. 209–211.

[ACJ14] Y. Adulyasak, J.-F. Cordeau, and R. Jans. "Optimization-Based Adaptive Large Neighborhood Search for the Production Routing Problem". In: *Transportation science* 48.1 (2014), pp. 20–45.

[AFG01] N. Ascheuer, M. Fischetti, and M. Grötschel. "Solving the Asymmetric Travelling Salesman Problem with Time Windows by Branch-and-Cut". In: *Mathematical Programming* 90.3 (May 2001), pp. 475–506. ISSN: 1436-4646. DOI: 10.1007/PL00011432.

[AGP14] N. Azi, M. Gendreau, and J.-Y. Potvin. "An Adaptive Large Neighborhood Search for a Vehicle Routing Problem with Multiple Routes". In: *Computers & Operations Research* 41 (2014), pp. 167–173.

[All+14] D. Allouche, I. André, S. Barbe, J. Davies, S. de Givry, G. Katsirelos, B. O'Sullivan, S. Prestwich, T. Schiex, and S. Traoré. "Computational Protein Design as an Optimization Problem". In: *Artificial Intelligence* 212 (2014), pp. 59–79.

[AM07] L. Art and H. Mauch. *Dynamic Programming: A Computational Tool*. Springer, 2007.

[Ant+20] V. Antuori, E. Hebrard, M.-J. Huguet, S. Essodaigui, and A. Nguyen. "Leveraging Reinforcement Learning, Constraint Programming and Local Search: A Case Study in Car Manufacturing". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 2020, pp. 657–672.

[App+11] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem*. Princeton university press, 2011.

[Apt03] K. Apt. "Local Consitency Notions". In: *Principles of Constraint Programming*. Cambridge University Press, Aug. 2003, pp. 135–177. ISBN: 978-0-521-82583-2.

[Art+14] C. Artigues, E. Hebrard, V. Mayer-Eichberger, M. Siala, and T. Walsh. "SAT and Hybrid Models of the Car Sequencing Problem". In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2014, pp. 268–283.

[Att+04] A. Attanasio, J.-F. Cordeau, G. Ghiani, and G. Laporte. "Parallel Tabu Search Heuristics for the Dynamic Multi-Vehicle Dial-a-Ride Problem". In: *Parallel Computing* 30.3 (2004), pp. 377–387.

[Aud+20] G. Audemard, F. Boussemart, C. Lecoutre, C. Piette, and O. Roussel. "XCSP3 and Its Ecosystem". In: *Constraints* 25.1 (Apr. 2020), pp. 47–69. ISSN: 1572-9354. DOI: 10.1007/s10601-019-09307-9.

[Bac+00] B. D. Backer, V. Furnon, P. Shaw, P. Kilby, and P. Prosser. "Solving Vehicle Routing Problems Using Constraint Programming and Metaheuristics". In: *Journal of Heuristics* 6.4 (Sept. 2000), pp. 501–523. ISSN: 1572-9397. DOI: 10.1023/A:1009621410177.

[Bar+07] A. Barra, L. Carvalho, N. Teypaz, V.-D. Cung, and R. Balassiano. "Solving the Transit Network Design Problem with Constraint Programming". In: *11th World Conference in Transport Research - WCTR 2007*. Berkeley, United States, June 2007.

[BB19] K. E. Booth and J. C. Beck. "A Constraint Programming Approach to Electric Vehicle Routing with Time Windows". In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2019, pp. 129–145.

[BCL12] G. Berbeglia, J.-F. Cordeau, and G. Laporte. "A Hybrid Tabu Search and Constraint Programming Algorithm for the Dynamic Dial-a-Ride Problem". In: *INFORMS Journal on Computing* 24.3 (2012), pp. 343–355.

[BCR12] N. Beldiceanu, M. Carlsson, and J.-X. Rampon. *Global Constraint Catalog, (Revision a)*. 2012.

[Bel+11] N. Beldiceanu, M. Carlsson, S. Demassey, and E. Poder. "New Filtering for the Cumulative Constraint in the Context of Non-Overlapping Rectangles". In: *Annals of Operations Research* 184.1 (Apr. 2011), pp. 27–50. ISSN: 1572-9338. DOI: 10.1007/s10479-010-0731-0.

[Bel14a] N. Beldiceanu. *Alldifferent Constraint*. https://sofdem.github.io/gccat/gccat/Calldifferent_cst.html. June 2014.

[Bel14b] N. Beldiceanu. *Circuit Constraint*. https://sofdem.github.io/gccat/gccat/Ccircuit.html. June 2014.

[Bel14c] N. Beldiceanu. *Cumulative Constraint*. https://sofdem.github.io/gccat/gccat/Ccumulative.html. June 2014.

[Bel14d] N. Beldiceanu. *Element Constraint*. https://sofdem.github.io/gccat/gccat/Celement.html. June 2014.

[Bel14e] N. Beldiceanu. *Precedence Constraint*. https://sofdem.github.io/gccat/gccat/Cprecedence.html. June 2014.

[Bel14f] N. Beldiceanu. *Sum Constraint*. https://sofdem.github.io/gccat/gccat/Csum_ctr.html. June 2014.

[Ber+07] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. "Static Pickup and Delivery Problems: A Classification Scheme and Survey". In: *Top* 15.1 (2007), pp. 1–31.

[Ber+14] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. "Optimization Bounds from Binary Decision Diagrams". In: *INFORMS Journal on Computing* 26.2 (2014), pp. 253–268.

[Bes+16] C. Bessiere, L. De Raedt, L. Kotthoff, S. Nijssen, B. O'Sullivan, and D. Pedreschi. *Data Mining and Constraint Programming*. Springer, 2016.

[BFW11] J. C. Beck, T. K. Feng, and J.-P. Watson. "Combining Constraint Programming and Local Search for Job-Shop Scheduling". In: *INFORMS Journal on Computing* 23.1 (2011), pp. 1–14.

[BG05a] O. Bräysy and M. Gendreau. "Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms". In: *Transportation Science* 39.1 (Feb. 2005), pp. 104–118. ISSN: 0041-1655. DOI: 10.1287/trsc.1030.0056.

[BG05b] O. Bräysy and M. Gendreau. "Vehicle Routing Problem with Time Windows, Part II: Metaheuristics". In: *Transportation Science* 39.1 (Feb. 2005), pp. 119–139. ISSN: 0041-1655. DOI: 10.1287/trsc.1030.0057.

[BH06] R. Bent and P. V. Hentenryck. "A Two-Stage Hybrid Algorithm for Pickup and Delivery Vehicle Routing Problems with Time Windows". In: *Computers & Operations Research*. Part Special Issue: Optimization Days 2003 33.4 (Apr. 2006), pp. 875–893. ISSN: 0305-0548. DOI: 10.1016/j.cor.2004.08.001.

[BHH11] D. Bergman, W.-J. van Hoeve, and J. N. Hooker. "Manipulating MDD Relaxations for Combinatorial Optimization". In: *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer, 2011, pp. 20–35.

[BHv09] A. Biere, M. Heule, and H. van Maaren. *Handbook of Satisfiability*. Vol. 185. IOS press, 2009.

[BK08] P. Barahona and L. Krippahl. "Constraint Programming in Structural Bioinformatics". In: *Constraints* 13.1 (June 2008), pp. 3–20. ISSN: 1572-9354. DOI: 10.1007/s10601-007-9036-6.

[BKS98] J. W. Baugh Jr, G. K. R. Kakivaya, and J. R. Stone. "Intractability of the Dial-a-Ride Problem and a Multiobjective Solution Using Simulated Annealing". In: *Engineering Optimization* 30.2 (1998), pp. 91–123.

[BLP21] Y. Bengio, A. Lodi, and A. Prouvost. "Machine Learning for Combinatorial Optimization: A Methodological Tour d'horizon". In: *European Journal of Operational Research* 290.2 (Apr. 2021), pp. 405–421. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2020.07.063.

[Boc+21] M. Boccia, A. Masone, A. Sforza, and C. Sterle. "A Column-and-Row Generation Approach for the Flying Sidekick Travelling Salesman Problem". In: *Transportation Research Part C: Emerging Technologies* 124 (Mar. 2021), p. 102913. ISSN: 0968-090X. DOI: 10.1016/j.trc.2020.102913.

[Bon17] G. Bonaccorso. *Machine Learning Algorithms*. Packt Publishing Ltd, 2017.

[Bou+04] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. "Boosting Systematic Search by Weighting Constraints". In: *Proceedings of the 16th European Conference on Artificial Intelligence*. 2004, pp. 146–150.

[BP14] P. Baudiš and P. Pošík. "Online Black-Box Algorithm Portfolios for Continuous Optimization". In: *Parallel Problem Solving from Nature – PPSN XIII*. Ed. by T. Bartz-Beielstein, J. Branke, B. Filipič, and J. Smith. Lecture Notes in Computer Science. Cham:

Springer International Publishing, 2014, pp. 40–49. ISBN: 978-3-319-10762-2. DOI: 10.1007/978-3-319-10762-2_4.

[BPA01] A. Bockmayr, N. Pisaruk, and A. Aggoun. "Network Flow Problems in Constraint Programming". In: *Principles and Practice of Constraint Programming — CP 2001*. Ed. by T. Walsh. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2001, pp. 196–210. ISBN: 978-3-540-45578-3. DOI: 10.1007/3-540-45578-7_14.

[BPR11] G. Berbeglia, G. Pesant, and L.-M. Rousseau. "Checking the Feasibility of Dial-a-Ride Instances Using Constraint Programming". In: *Transportation Science* 45.3 (2011), pp. 399–412.

[Bra22] K. Braekers. *Dial-a-Ride Problems Instances*. http://alpha.uhasselt.be/kris.braekers/. Oct. 2022.

[BRV16] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse. "The Vehicle Routing Problem: State of the Art Classification and Review". In: *Computers & Industrial Engineering* 99 (Sept. 2016), pp. 300–313. ISSN: 0360-8352. DOI: 10.1016/j.cie.2015.12.007.

[BS12] N. Beldiceanu and H. Simonis. "A Model Seeker: Extracting Global Constraint Models from Positive Examples". In: *Principles and Practice of Constraint Programming*. 2012, pp. 141–157.

[Cap+21] Q. Cappart, T. Moisan, L.-M. Rousseau, I. Prémont-Schwarz, and A. A. Cire. "Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. May 2021, pp. 3677–3687. DOI: 10.1609/aaai.v35i5.16484.

[CB09] T. Carchrae and J. C. Beck. "Principles for the Design of Large Neighborhood Search". In: *Journal of Mathematical Modelling and Algorithms* 8.3 (2009), pp. 245–270.

[CB89] N. Christofides and E. Benavent. "An Exact Algorithm for the Quadratic Assignment Problem on a Tree". In: *Operations Research* 37.5 (1989), pp. 760–768.

[CGL97] J.-F. Cordeau, M. Gendreau, and G. Laporte. "A Tabu Search Heuristic for Periodic and Multi-Depot Vehicle Routing Problems". In: *Networks* 30.2 (1997), pp. 105–119.

[Cha+21] F. Chalumeau, I. Coulon, Q. Cappart, and L.-M. Rousseau. "Seapearl: A Constraint Programming Solver Guided by Reinforcement Learning". In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2021, pp. 392–409.

[CHN01] C. W. Choi, M. Henz, and K. B. Ng. "Components for State Restoration in Tree Search". In: *Principles and Practice of Constraint Programming — CP 2001*. Ed. by T. Walsh. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2001, pp. 240–255. ISBN: 978-3-540-45578-3. DOI: 10.1007/3-540-45578-7_17.

[CL03a] J.-F. Cordeau and G. Laporte. "A Tabu Search Heuristic for the Static Multi-Vehicle Dial-a-Ride Problem". In: *Transportation Research Part B: Methodological* 37.6 (2003), pp. 579–594.

[CL03b] J.-F. Cordeau and G. Laporte. "The Dial-a-Ride Problem (DARP): Variants, Modeling Issues and Algorithms". In: *4OR* 1.2 (2003), pp. 89–101. DOI: 10.1007/s10288-002-0009-8.

[CL07] J.-F. Cordeau and G. Laporte. "The Dial-a-Ride Problem: Models and Algorithms". In: *Annals of Operations Research* 153.1 (Sept. 2007), pp. 29–46. ISSN: 1572-9338. DOI: 10.1007/s10479-007-0170-8.

[CL96] Y. Caseau and F. Laburthe. "Cumulative Scheduling with Task Intervals". In: *JICSLP*. Vol. 96. 1996, pp. 369–383.

[Cor+05] J.-F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, and J.-S. Sormany. "New Heuristics for the Vehicle Routing Problem". In: *Logistics systems: design and optimization* (2005), pp. 279–297.

[Cor06] J.-F. Cordeau. "A Branch-and-Cut Algorithm for the Dial-a-Ride Problem". In: *Operations Research* 54.3 (2006), pp. 573–586.

[CP80] H. Crowder and M. W. Padberg. "Solving Large-Scale Symmetric Travelling Salesman Problems to Optimality". In: *Management Science* 26.5 (May 1980), pp. 495–509. ISSN: 0025-1909. DOI: 10.1287/mnsc.26.5.495.

[CS15] G. Chu and P. J. Stuckey. "Learning Value Heuristics for Constraint Programming". In: *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. 2015, pp. 108–123.

[CSD] CSD Liège. *Centrale de services à domicile*. https://www.csdliege.be/fr.

[DAA21] R. Durgut, M. E. Aydin, and I. Atli. "Adaptive Operator Selection with Reinforcement Learning". In: *Information Sciences* 581 (2021), pp. 773–790.

[Dav10] D. Davendra. *Traveling Salesman Problem: Theory and Applications*. BoD–Books on Demand, 2010.

[DBL12] E. Demir, T. Bektaş, and G. Laporte. "An Adaptive Large Neighborhood Search Heuristic for the Pollution-Routing Problem". In: *European Journal of Operational Research* 223.2 (2012), pp. 346–359.

[DD92] K. A. Dowsland and W. B. Dowsland. "Packing Problems". In: *European Journal of Operational Research* 56.1 (Jan. 1992), pp. 2–14. ISSN: 0377-2217. DOI: 10.1016/0377-2217(92)90288-K.

[DDD05] G. Dooms, Y. Deville, and P. Dupont. "Cp (Graph): Introducing a Graph Computation Domain in Constraint Programming". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 2005, pp. 211–225.

[DDS06] G. Desaulniers, J. Desrosiers, and M. M. Solomon. *Column Generation*. Vol. 5. Springer Science & Business Media, 2006.

[de +13] V. l. C. de Saint-Marcq, P. Schaus, C. Solnon, and C. Lecoutre. "Sparse-Sets for Domain Implementation". In: *CP Workshop on Techniques foR Implementing Constraint Programming Systems (TRICS)*. 2013, pp. 1–10.

[DGN10] L. De Raedt, T. Guns, and S. Nijssen. "Constraint Programming for Data Mining and Machine Learning". In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010.

[DK11] D.-Z. Du and K.-I. Ko. *Theory of Computational Complexity*. Vol. 58. John Wiley & Sons, 2011.

[DPd17] P. Detti, F. Papalini, and G. Z. M. de Lara. "A Multi-Depot Dial-a-Ride Problem with Heterogeneous Vehicles and Compatibility Constraints in Healthcare". In: *Omega* 70 (2017), pp. 1–14.

[DPR06] S. Demassey, G. Pesant, and L.-M. Rousseau. "A Cost-Regular Based Hybrid Column Generation Approach". In: *Constraints* 11.4 (2006), pp. 315–333.

[DSV22] A. Delecluse, P. Schaus, and P. Van Hentenryck. "Sequence Variables for Routing Problems". In: *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*. 2022.

[DT03] G. Dantzig and M. N. Thapa. *Linear Programming 2: Theory and Extensions*. Springer Series in Operations Research and Financial Engineering. New York: Springer-Verlag, 2003. ISBN: 978-0-387-98613-5. DOI: 10.1007/b97283.

[Dum+95] Y. Dumas, J. Desrosiers, E. Gelinas, and M. M. Solomon. "An Optimal Algorithm for the Traveling Salesman Problem with Time Windows". In: *Operations Research* 43.2 (Apr. 1995), pp. 367–371. ISSN: 0030-364X. DOI: 10.1287/opre.43.2.367.

[DVS15] C. Dejemeppe, S. Van Cauwelaert, and P. Schaus. "The Unary Resource with Transition Times". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 2015, pp. 89–104.

[Enc19] Encik Tekateki. *Solution J for 8 Queen Puzzles*. https://commons.wikimedia.org/w/index.php?curid=77832464. Apr. 2019.

[EVR09] B. Eksioglu, A. V. Vural, and A. Reisman. "The Vehicle Routing Problem: A Taxonomic Review". In: *Computers & Industrial Engineering* 57.4 (Nov. 2009), pp. 1472–1483. ISSN: 0360-8352. DOI: 10.1016/j.cie.2009.05.009.

[FD94] D. Frost and R. Dechter. "In Search of the Best Constraint Satisfaction Search". In: *AAAI*. 1994.

[Fle90] B. Fleischmann. "The Discrete Lot-Sizing and Scheduling Problem". In: *European Journal of Operational Research* 44.3 (1990), pp. 337–348.

[Fou+17] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas. "A Survey of Sequential Pattern Mining". In: *Data Science and Pattern Recognition* 1.1 (2017), pp. 54–77.

[FS14] K. G. Francis and P. J. Stuckey. "Explaining Circuit Propagation". In: *Constraints* 19.1 (2014), pp. 1–29.

[Gar+10] T. Garaix, C. Artigues, D. Feillet, and D. Josselin. "Vehicle Routing Problems with Alternative Paths: An Application to on-Demand Transportation". In: *European Journal of Operational Research* 204.1 (July 2010), pp. 62–75. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2009.10.002.

[Gay+15] S. Gay, R. Hartert, C. Lecoutre, and P. Schaus. "Conflict Ordering Search for Scheduling Problems". In: *International Conference on Principles and Practice of Constraint Programming*. 2015, pp. 140–148.

[GBY01] D. Gilbert, R. Backofen, and R. H. Yap. "Introduction to the Special Issue on Bioinformatics". In: *Constraints* 6.2/3 (2001), p. 139.

[GD19] T. Gschwind and M. Drexl. "Adaptive Large Neighborhood Search with a Constant-Time Feasibility Test for the Dial-a-Ride Problem". In: *Transportation Science* 53.2 (Mar. 2019), pp. 480–491. ISSN: 0041-1655. DOI: 10.1287/trsc.2018.0837.

[Ger94] C. Gervet. "Conjunto: Constraint Logic Programming with Finite Set Domains". In: *ILPS*. 1994.

[Ger97] C. Gervet. "Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language". In: *Constraints* 1.3 (1997), pp. 191–244.

[GHS15a] S. Gay, R. Hartert, and P. Schaus. "Simple and Scalable Time-Table Filtering for the Cumulative Constraint". In: *International Conference on Principles and Practice of Constraint Programming*. 2015, pp. 149–157.

[GHS15b] S. Gay, R. Hartert, and P. Schaus. "Time-Table Disjunctive Reasoning for the Cumulative Constraint". In: *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. 2015, pp. 157–172.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GLN05] D. Godard, P. Laborie, and W. Nuijten. "Randomized Large Neighborhood Search for Cumulative Scheduling." In: *ICAPS*. Vol. 5. 2005, pp. 81–89.

[GP10] M. Gendreau and J.-Y. Potvin. *Handbook of Metaheuristics*. Vol. 2. Springer, 2010.

[Gre08] F. Greco. *Traveling Salesman Problem*. BoD–Books on Demand, 2008.

[GRW08] B. L. Golden, S. Raghavan, and E. A. Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer Science & Business Media, July 2008. ISBN: 978-0-387-77778-8.

[GSC21] X. Gillard, P. Schaus, and V. Coppé. "Ddo, a Generic and Efficient Framework for MDD-based Optimization". In: *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*. 2021, pp. 5243–5245.

[GSD14] S. Gay, P. Schaus, and V. De Smedt. "Continuous Casting Scheduling with Constraint Programming". In: *International Conference on Principles and Practice of Constraint Programming*. 2014, pp. 831–845.

[GV06] C. Gervet and P. Van Hentenryck. "Length-Lex Ordering for Set Csps". In: *AAAI*. 2006, pp. 48–53.

[GW95] T. Gau and G. Wäscher. "CUTGEN1: A Problem Generator for the Standard One-Dimensional Cutting Stock Problem". In: *European Journal of Operational Research*. Cutting and Packing 84.3 (Aug. 1995), pp. 572–579. ISSN: 0377-2217. DOI: 10.1016/0377-2217(95)00023-J.

[Har+15a] R. Hartert, P. Schaus, S. Vissicchio, and O. Bonaventure. "Solving Segment Routing Problems with Hybrid Constraint Programming Techniques". In: *Principles and Practice of Constraint Programming*. Ed. by G. Pesant. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 592–608. ISBN: 978-3-319-23219-5. DOI: 10.1007/978-3-319-23219-5_41.

[Har+15b] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois. "A Declarative and Expressive Approach to Control Forwarding Paths in Carrier-Grade Networks". In: *ACM SIGCOMM computer communication review* 45.4 (2015), pp. 15–28.

[HB12] S. Heinz and J. C. Beck. "Reconsidering Mixed Integer Programming and MIP-Based Hybrids for Scheduling". In: *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*. Ed. by N. Beldiceanu, N. Jussien, and É. Pinson. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 211–227. ISBN: 978-3-642-29828-8. DOI: 10.1007/978-3-642-29828-8_14.

[He+19] Y. He, S. Y. Yuen, Y. Lou, and X. Zhang. "A Sequential Algorithm Portfolio Approach for Black Box Optimization". In: *Swarm and Evolutionary Computation* 44 (Feb. 2019), pp. 559–570. ISSN: 2210-6502. DOI: 10.1016/j.swevo.2018.07.001.

[Hen22] G. Hendel. "Adaptive Large Neighborhood Search for Mixed Integer Programming". In: *Mathematical Programming Computation* 14.2 (June 2022), pp. 185–221. ISSN: 1867-2957. DOI: 10.1007/s12532-021-00209-7.

[HK13] E. Huang and R. E. Korf. "Optimal Rectangle Packing: An Absolute Placement Approach". In: *Journal of Artificial Intelligence Research* 46 (2013), pp. 47–87.

[HM98] W. He and K. Marriott. "Constrained Graph Layout". In: *Constraints* 3.4 (1998), pp. 289–314.

[Hoo12] J. N. Hooker. *Integrated Methods for Optimization*. Vol. 170. International Series in Operations Research & Management Science. Boston, MA: Springer US, 2012. ISBN: 978-1-4614-1899-3 978-1-4614-1900-6. DOI: 10.1007/978-1-4614-1900-6.

[Hou+14] V. R. Houndji, P. Schaus, L. Wolsey, and Y. Deville. "The Stockingcost Constraint". In: *International Conference on Principles and Practice of Constraint Programming*. 2014, pp. 382–397.

[HPR13] K. L. Hoffman, M. Padberg, and G. Rinaldi. "Traveling Salesman Problem". In: *Encyclopedia of Operations Research and Management Science*. Ed. by S. I. Gass and M. C. Fu. Boston, MA: Springer US, 2013, pp. 1573–1578. ISBN: 978-1-4419-1153-7. DOI: 10.1007/978-1-4419-1153-7_1068.

[HS17] E. Hebrard and M. Siala. "Explanation-Based Weighted Degree". In: *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. 2017, pp. 167–175.

[IBM19a] IBM Knowledge Center. *Interval Variable Sequencing in CP Optimizer*. https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.ide.help/refcppopl/html/interval_sequence.html. 2019.

[IBM19b] IBM Knowledge Center. *Search API for Scheduling in CP Optimizer*. https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.cpo.help/refcppcpoptimizer/html/sched_search_api.html. 2019.

[IBM21] IBM Knowledge Center. *Interval Variables*. https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/icos/20.1.0?topic=scheduling-interval-variables. Nov. 2021.

[IJ14] K. Ilavarasi and K. S. Joseph. "Variants of Travelling Salesman Problem: A Survey". In: *International Conference on Information Communication and Embedded Systems (ICICES2014)*. Feb. 2014, pp. 1–7. DOI: 10.1109/ICICES.2014.7033850.

[ITV14] S. Irnich, P. Toth, and D. Vigo. "Chapter 1: The Family of Vehicle Routing Problems". In: *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. SIAM, 2014, pp. 1–33.

[JB04] R. K. Jana and M. P. Biswal. "Stochastic Simulation-Based Genetic Algorithm for Chance Constraint Programming Problems with Continuous Random Variables". In: *International Journal of Computer Mathematics* 81.9 (2004), pp. 1069–1076.

[JK89] B. Jeromin and F. Körner. "Triangle Inequality and Symmetry in Connection with the Assignment and the Traveling Salesman Problem". In: *European Journal of Operational Research* 38.1 (Jan. 1989), pp. 70–75. ISSN: 0377-2217. DOI: 10.1016/0377-2217(89)90470-0.

[JST08] N. Jozefowiez, F. Semet, and E.-G. Talbi. "Multi-Objective Vehicle Routing Problems". In: *European Journal of Operational Research* 189.2 (Sept. 2008), pp. 293–309. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2007.05.055.

[JV11] S. Jain and P. Van Hentenryck. "Large Neighborhood Search for Dial-a-Ride Problems". In: *Principles and Practice of Constraint Programming–CP 2011* (2011), pp. 400–413.

[Kam+14] R. Kameugne, L. P. Fotso, J. Scott, and Y. Ngo-Kateu. "A Quadratic Edge-Finding Filtering Algorithm for Cumulative Resource Constraints". In: *Constraints* 19.3 (July 2014), pp. 243–269. ISSN: 1572-9354. DOI: 10.1007/s10601-013-9157-z.

[Kar72] R. M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations*. Springer, 1972, pp. 85–103.

[Kar75] R. M. Karp. "On the Computational Complexity of Combinatorial Problems". In: *Networks* 5.1 (1975), pp. 45–68. ISSN: 1097-0037. DOI: 10.1002/net.1975.5.1.45.

[KFM71] P. Krolak, W. Felts, and G. Marble. "A Man-Machine Approach toward Solving the Traveling Salesman Problem". In: *Communications of the ACM* 14.5 (1971), pp. 327–334.

[KGM12] L. Kotthoff, I. P. Gent, and I. Miguel. "An Evaluation of Machine Learning in Algorithm Selection for Search Problems". In: *AI Communications* 25.3 (Jan. 2012), pp. 257–270. ISSN: 0921-7126. DOI: 10.3233/AIC-2012-0533.

[KHN04] K. Katayama, A. Hamamoto, and H. Narihisa. "Solving the Maximum Clique Problem by K-Opt Local Search". In: *Proceedings of the 2004 ACM Symposium on Applied Computing*. SAC '04. New York, NY, USA: Association for Computing Machinery, Mar. 2004, pp. 1021–1025. ISBN: 978-1-58113-812-2. DOI: 10.1145/967900.968107.

[KLT20] Ç. Koç, G. Laporte, and İ. Tükenmez. "A Review of Vehicle Routing with Simultaneous Pickup and Delivery". In: *Computers & Operations Research* 122 (Oct. 2020), p. 104987. ISSN: 0305-0548. DOI: 10.1016/j.cor.2020.104987.

[Kot10] L. Kotthoff. *Constraint Solvers: An Empirical Evaluation of Design Decisions*. Jan. 2010. DOI: 10.48550/arXiv.1002.0134. arXiv: 1002.0134 [cs].

[Kov+12] A. A. Kovacs, S. N. Parragh, K. F. Doerner, and R. F. Hartl. "Adaptive Large Neighborhood Search for Service Technician Routing and Scheduling Problems". In: *Journal of scheduling* 15.5 (2012), pp. 579–600.

[KP14] V. Kuleshov and D. Precup. "Algorithms for Multi-Armed Bandit Problems". In: *arXiv preprint arXiv:1402.6028* (2014). arXiv: 1402.6028.

[Kru56] J. B. Kruskal. "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem". In: *Proceedings of the American Mathematical society* 7.1 (1956), pp. 48–50.

[KS99] D. L. Kreher and D. R. Stinson. "Combinatorial Algorithms: Generation, Enumeration, and Search". In: *ACM SIGACT News* 30.1 (1999), pp. 33–35.

[KSN07] K. Katayama, M. Sadamatsu, and H. Narihisa. "Iterated K-Opt Local Search for the Maximum Clique Problem". In: *Evolutionary Computation in Combinatorial Optimization*. Ed. by C. Cotta and J. van Hemert. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 84–95. ISBN: 978-3-540-71615-0. DOI: 10.1007/978-3-540-71615-0_8.

[Lab+09] P. Laborie, J. Rogerie, P. Shaw, and P. Vil\'\im. "Reasoning with Conditional Time-Intervals. Part II: An Algebraical Model for Resources." In: *FLAIRS Conference*. 2009, pp. 201–206.

[Lab+18] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. "IBM ILOG CP Optimizer for Scheduling: 20+ Years of Scheduling with Constraints at IBM/ILOG". In: *Constraints* 23 (Oct. 2018). DOI: 10.1007/s10601-018-9281-x.

[LAB18] C. Liu, D. M. Aleman, and J. C. Beck. "Modelling and Solving the Senior Transportation Problem". In: *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2018, pp. 412–428.

[Lab18a] P. Laborie. "An Update on the Comparison of MIP, CP and Hybrid Approaches for Mixed Resource Allocation and Scheduling". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Ed. by W.-J. van Hoeve. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 403–411. ISBN: 978-3-319-93031-2. DOI: 10.1007/978-3-319-93031-2_29.

[Lab18b] P. Laborie. "Objective Landscapes for Constraint Programming". In: *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. 2018, pp. 387–402.

[Lap92] G. Laporte. "The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms". In: *European Journal of Operational Research* 59.3 (June 1992), pp. 345–358. ISSN: 0377-2217. DOI: 10.1016/0377-2217(92)90192-C.

[Lau78] J.-L. Lauriere. "A Language and a Program for Stating and Solving Combinatorial Problems". In: *Artificial intelligence* 10.1 (1978), pp. 29–127.

[Law84] S. Lawrence. "Resouce Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques". In: *Graduate School of Industrial Administration, Carnegie-Mellon University* (1984).

[LCP22] D. Lafleur, S. Chandar, and G. Pesant. "Combining Reinforcement Learning and Constraint Programming for Sequence-Generation Tasks with Hard Constraints". In: *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[Lec96] M. Leconte. "A Bounds-Based Reduction Scheme for Constraints of Difference". In: *Proceedings of the Constraint-96 International Workshop on Constraint-Based Reasoning*. 1996, pp. 19–28.

[Lew16] R. Lewis. *A Guide to Graph Colouring*. Cham: Springer International Publishing, 2016. ISBN: 978-3-319-25728-0 978-3-319-25730-3. DOI: 10.1007/978-3-319-25730-3.

[LG07] P. Laborie and D. Godard. "Self-Adapting Large Neighborhood Search: Application to Single-Mode Scheduling Problems". In: *Proceedings MISTA-07, Paris* 8 (2007).

[Li+22] H. Li, Y. Wu, M. Yin, and Z. Li. "A Portfolio-Based Approach to Select Efficient Variable Ordering Heuristics for Constraint Satisfaction Problems". In: *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[Liu+13] R. Liu, X. Xie, V. Augusto, and C. Rodriguez. "Heuristic Algorithms for a Vehicle Routing Problem with Simultaneous Delivery and Pickup and Time Windows in Home Health Care". In: *European Journal of Operational Research* 230.3 (2013), pp. 475–486.

[LK75] J. K. Lenstra and A. H. G. R. Kan. "Some Simple Applications of the Travelling Salesman Problem". In: *Journal of the Operational Research Society* 26.4 (Dec. 1975), pp. 717–733. ISSN: 0160-5682. DOI: 10.1057/jors.1975.151.

[Log] Logistics in Wallonia. *PRESupply*. https://www.logisticsinwallonia.be/fr/projets/presupply.

[Lóp+03] A. López-Ortiz, C.-G. Quimper, J. Tromp, and P. Van Beek. "A Fast and Simple Algorithm for Bounds Consistency of the Alldifferent Constraint". In: *IJCAI*. Vol. 3. 2003, pp. 245–250.

[Lóp+13] M. López-Ibáñez, C. Blum, J. W. Ohlmann, and B. W. Thomas. "The Travelling Salesman Problem with Time Windows: Adapting Algorithms from Travel-Time to Makespan Optimization". In: *Applied Soft Computing* 13.9 (Sept. 2013), pp. 3806–3815. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2013.05.009.

[Lot+13] M. Loth, M. Sebag, Y. Hamadi, and M. Schoenauer. "Bandit-Based Search for Constraint Programming". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 2013, pp. 464–480.

[LR08] P. Laborie and J. Rogerie. "Reasoning with Conditional Time-Intervals." In: *FLAIRS Conference*. 2008, pp. 555–560.

[LS14] M. Lombardi and P. Schaus. "Cost Impact Guided LNS". In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2014, pp. 293–300.

[LSD90] A. Langevin, F. Soumis, and J. Desrosiers. "Classification of Travelling Salesman Problem Formulations". In: *Operations Research Letters* 9.2 (Mar. 1990), pp. 127–132. ISSN: 0167-6377. DOI: 10.1016/0167-6377(90)90052-7.

[Mal+13] Y. Malitsky, D. Mehta, B. O'Sullivan, and H. Simonis. "Tuning Parameters of Large Neighborhood Search for the Machine Reassignment Problem". In: *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. 2013, pp. 176–192.

[Mal+18] C. Maleck, G. Nieke, K. Bock, D. Pabst, and M. Stehli. "A Comparison of a CP and MIP Approach for Scheduling Jobs in Production Areas with Time Constraints and Uncertainties". In: *2018 Winter Simulation Conference (WSC)*. Dec. 2018, pp. 3526–3537. DOI: 10.1109/WSC.2018.8632404.

[Maz+21] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev. "Reinforcement Learning for Combinatorial Optimization: A Survey". In: *Computers & Operations Research* 134 (Oct. 2021), p. 105400. ISSN: 0305-0548. DOI: 10.1016/j.cor.2021.105400.

[MBC17] Y. Molenbruch, K. Braekers, and A. Caris. "Typology and Literature Review for Dial-a-Ride Problems". In: *Annals of Operations Research* 259.1 (Dec. 2017), pp. 295–325. ISSN: 1572-9338. DOI: 10.1007/s10479-017-2525-0.

[MDD07] J.-N. Monette, Y. Deville, and P. Dupont. "A Position-Based Propagator for the Open-Shop Problem". In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer, 2007, pp. 186–199.

[MDV09] J.-N. Monette, Y. Deville, and P. Van Hentenryck. "Aeon: Synthesizing Scheduling Algorithms from High-Level Models". In: *Operations Research and Cyber-Infrastructure*. Springer, 2009, pp. 43–59.

[MDV11] J.-B. Mairy, Y. Deville, and P. Van Hentenryck. "Reinforced Adaptive Large Neighborhood Search". In: *The Seventeenth International Conference on Principles and Practice of Constraint Programming (CP 2011)*. Springer Berlin/Heidelberg, Germany, 2011, p. 55.

[MG07] J. Matoušek and B. Gärtner. *Understanding and Using Linear Programming*. Vol. 33. Springer, 2007.

[Mig] I. Miguel. *CSPLib Problem 038: Steel Mill Slab Design*. http://www.csplib.org/Problems/prob038.

[MIM07] E. Melachrinoudis, A. B. Ilhan, and H. Min. "A Dial-a-Ride Problem for Client Transportation in a Health-Care Organization". In: *Computers & Operations Research* 34.3 (2007), pp. 742–759.

[Mina] MiniZinc Team. *MiniZinc*. https://www.minizinc.org/.

[Minb] MiniZinc Team. *The MiniZinc Handbook — The MiniZinc Handbook 2.6.4*. https://www.minizinc.org/doc-2.6.4/en/index.html.

[MLP13] R. Masson, F. Lehuédé, and O. Péton. "An Adaptive Large Neighborhood Search for the Pickup and Delivery Problem with Transfers". In: *Transportation Science* 47.3 (2013), pp. 344–355.

[MM11] E. Melachrinoudis and H. Min. "A Tabu Search Heuristic for Solving the Multi-Depot, Multi-Vehicle, Double Request Dial-a-Ride Problem Faced by a Healthcare Organisation". In: *International Journal of Operational Research* 10.2 (2011), pp. 214–239.

[Mon+07] J.-N. Monette, P. Schaus, S. Zampelli, Y. Deville, P. Dupont, et al. "A CP Approach to the Balanced Academic Curriculum Problem". In: *Seventh International Workshop on Symmetry and Constraint Satisfaction Problems*. Vol. 7. 2007.

[Mon+15] J. R. Montoya-Torres, J. López Franco, S. Nieto Isaza, H. Felizzola Jiménez, and N. Herazo-Padilla. "A Literature Review on the Vehicle Routing Problem with Multiple Depots". In: *Computers & Industrial Engineering* 79 (Jan. 2015), pp. 115–129. ISSN: 0360-8352. DOI: 10.1016/j.cie.2014.10.029.

[MR22] R. Martí and G. Reinelt. *Exact and Heuristic Methods in Combinatorial Optimization*. Springer, 2022.

[MSD10] J.-B. Mairy, P. Schaus, and Y. Deville. "Generic Adaptive Heuristics for Large Neighborhood Search". In: *Seventh International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS2010). A Satellite Workshop of CP*. Citeseer, 2010.

[MSV21] L. Michel, P. Schaus, and P. Van Hentenryck. "MiniCP: A Lightweight Solver for Constraint Programming". In: *Mathematical Programming Computation* 13.1 (2021), pp. 133–184. DOI: 10.1007/s12532-020-00190-7.

[MT00] K. Mehlhorn and S. Thiel. "Faster Algorithms for Bound-Consistency of the Sortedness and the Alldifferent Constraint". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 2000, pp. 306–319.

[MT08] A. Mahajan and D. Teneketzis. "Multi-Armed Bandit Problems". In: *Foundations and Applications of Sensor Management*. Springer, 2008, pp. 121–151.

[MV04] L. Michel and P. Van Hentenryck. "A Simple Tabu Search for Warehouse Location". In: *European Journal of Operational Research* 157.3 (Sept. 2004), pp. 576–591. ISSN: 0377-2217. DOI: 10.1016/S0377-2217(03)00247-9.

[MV08] L. Mercier and P. Van Hentenryck. "Edge Finding for Cumulative Scheduling". In: *INFORMS Journal on Computing* 20.1 (Feb. 2008), pp. 143–153. ISSN: 1091-9856. DOI: 10.1287/ijoc.1070.0226.

[MV12] L. Michel and P. Van Hentenryck. "Activity-Based Search for Black-Box Constraint Programming Solvers". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimzation Problems* (2012), pp. 228–243.

[Näh11] S. Näher. "The Travelling Salesman Problem". In: *Algorithms Unplugged*. Springer, 2011, pp. 383–391.

[Nar03] A. Nareyek. "Choosing Search Heuristics by Non-Stationary Reinforcement Learning". In: *Metaheuristics: Computer Decision-Making*. Springer, 2003, pp. 523–544.

[Net+07] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. "MiniZinc: Towards a Standard CP Modelling Language". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 2007, pp. 529–543.

[NZ14] S. Nijssen and A. Zimmermann. "Constraint-Based Pattern Mining". In: *Frequent Pattern Mining*. Springer, 2014, pp. 147–163.

[NZE05] P. Ngatchou, A. Zarei, and A. El-Sharkawi. "Pareto Multi Objective Optimization". In: *Intelligent Systems Application to Power Systems, 2005. Proceedings of the 13th International Conference On*. 2005, pp. 84–91.

[ONZ21] J. B. Odili, A. Noraziah, and M. Zarina. "A Comparative Performance Analysis of Computational Intelligence Techniques to Solve the Asymmetric Travelling Salesman Problem". In: *Computational Intelligence and Neuroscience* 2021 (Apr. 2021), e6625438. ISSN: 1687-5265. DOI: 10.1155/2021/6625438.

[OQ13] P. Ouellet and C.-G. Quimper. "Time-Table Extended-Edge-Finding for the Cumulative Constraint". In: *International Conference on Principles and Practice of Constraint Programming*. 2013, pp. 562–577.

[OSC09] O. Ohrimenko, P. J. Stuckey, and M. Codish. "Propagation via Lazy Clause Generation". In: *Constraints* 14.3 (2009), pp. 357–391.

[Osc12] OscaR Team. *OscaR: Scala in OR*. https://bitbucket.org/oscarlib/oscar. 2012.

[PAM04] M. Palpant, C. Artigues, and P. Michelon. "LSSPER: Solving the Resource-Constrained Project Scheduling Problem with Large Neighbourhood Search". In: *Annals of Operations Research* 131.1 (Oct. 2004), pp. 237–257. ISSN: 1572-9338. DOI: 10.1023/B:ANOR.0000039521.26237.62.

[Par+09] S. N. Parragh, K. F. Doerner, R. F. Hartl, and X. Gandibleux. "A Heuristic Two-Phase Solution Approach for the Multi-Objective Dial-a-Ride Problem". In: *Networks* 54.4 (2009), pp. 227–242.

[Par+12] S. N. Parragh, J.-F. Cordeau, K. F. Doerner, and R. F. Hartl. "Models and Algorithms for the Heterogeneous Dial-a-Ride Problem with Driver-Related Constraints". In: *OR spectrum* 34.3 (2012), pp. 593–633.

[Par11] S. N. Parragh. "Introducing Heterogeneous Users and Vehicles into Models and Algorithms for the Dial-a-Ride Problem". In: *Transportation Research Part C: Emerging Technologies* 19.5 (2011), pp. 912–930.

[PCP20] C. Paquay, Y. Crama, and T. Pironet. "Recovery Management for a Dial-a-Ride System with Real-Time Disruptions". In: *European Journal of Operational Research* 280.3 (2020), pp. 953–969.

[PF19a] L. Perron and V. Furnon. *OR-Tools*. https://developers.google.com/optimization/. 2019.

[PF19b] L. Perron and V. Furnon. *OR-Tools Sequence Var*. https://developers.google.com/optimization/reference/constraint_solver/constraint_solver/SequenceVar. 2019.

[PH11] D. Pacino and P. V. Hentenryck. "Large Neighborhood Search and Adaptive Randomized Decompositions for Flexible Jobshop Scheduling". In: *Twenty-Second International Joint Conference on Artificial Intelligence*. https://www.aaai.org/ocs/index.php/IJCAI/IJCAI11/paper/view/3062. June 2011.

[PHW07] J. Pei, J. Han, and W. Wang. "Constraint-Based Sequential Pattern Mining: The Pattern-Growth Methods". In: *Journal of Intelligent Information Systems* 28.2 (2007), pp. 133–160.

[Pic+17] É. Picard-Cantin, M. Bouchard, C.-G. Quimper, and J. Sweeney. "Learning the Parameters of Global Constraints Using Branch-and-Bound". In: *International Conference on Principles and Practice of Constraint Programming*. 2017, pp. 512–528.

[Pil+13] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia. "A Review of Dynamic Vehicle Routing Problems". In: *European Journal of Operational Research* 225.1 (Feb. 2013), pp. 1–11. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2012.08.015.

[PLJ14] C. Prud'homme, X. Lorca, and N. Jussien. "Explanation-Based Large Neighborhood Search". In: *Constraints* 19.4 (Oct. 2014), pp. 339–379. ISSN: 1572-9354. DOI: 10.1007/s10601-014-9166-6.

[PQZ12] G. Pesant, C.-G. Quimper, and A. Zanarini. "Counting-Based Search: Branching Heuristics for Constraint Satisfaction Problems". In: *Journal of Artificial Intelligence Research* (2012).

[PR07] D. Pisinger and S. Ropke. "A General Heuristic for Vehicle Routing Problems". In: *Computers & operations research* 34.8 (2007), pp. 2403–2435.

[PRS16] A. Palmieri, J.-C. Régin, and P. Schaus. "Parallel Strategies Selection". In: *International Conference on Principles and Practice of Constraint Programming*. 2016, pp. 388–404.

[PS13] S. N. Parragh and V. Schmid. "Hybrid Column Generation and Large Neighborhood Search for the Dial-a-Ride Problem". In: *Computers & Operations Research* 40.1 (2013), pp. 490–497.

[Psa83] H. N. Psaraftis. "An Exact Algorithm for the Single Vehicle Many-to-Many Dial-a-Ride Problem with Time Windows". In: *Transportation science* 17.3 (1983), pp. 351–357.

[PSF04] L. Perron, P. Shaw, and V. Furnon. "Propagation Guided Large Neighborhood Search". In: *Principles and Practice of Constraint Programming – CP 2004*. Ed. by M. Wallace. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 468–481. ISBN: 978-3-540-30201-8. DOI: 10.1007/978-3-540-30201-8_35.

[Pug04] J.-F. Puget. "Constraint Programming next Challenge: Simplicity of Use". In: *Principles and Practice of Constraint Programming–CP 2004* (2004), pp. 5–8.

[Pug98] J.-F. Puget. "A Fast Algorithm for the Bound Consistency of Alldiff Constraints". In: *Aaai/Iaai*. 1998, pp. 359–366.

[R197] J. Régin, J. P. .-. C. o. P. a. P. o. C. ..., and u. 1997. "A Filtering Algorithm for Global Sequencing Constraints". In: *Springer* 1330 (1997), pp. 32–46. DOI: 10.1007/BFb0017428.

[Ral+03] T. Ralphs, L. Kopman, W. Pulleyblank, and L. Trotter. "On the Capacitated Vehicle Routing Problem". In: *Mathematical Programming* 94.2 (Jan. 2003), pp. 343–359. ISSN: 1436-4646. DOI: 10.1007/s10107-002-0323-0.

[Red+22] M. Reda, A. Onsy, M. A. Elhosseini, A. Y. Haikal, and M. Badawy. "A Discrete Variant of Cuckoo Search Algorithm to Solve the Travelling Salesman Problem and Path Planning for Autonomous Trolley inside Warehouse". In: *Knowledge-Based Systems* 252 (Sept. 2022), p. 109290. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2022.109290.

[Ref04] P. Refalo. "Impact-Based Search Strategies for Constraint Programming". In: *CP* 3258 (2004), pp. 557–571.

[Rei+09] R. M. Reischuk, C. Schulte, P. J. Stuckey, and G. Tack. "Maintaining State in Propagation Solvers". In: *Principles and Practice of Constraint Programming - CP 2009*. Ed. by I. P. Gent. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 692–706. ISBN: 978-3-642-04244-7. DOI: 10.1007/978-3-642-04244-7_54.

[Rig20] R. Righi. *Scheduling Problems: New Applications and Trends*. BoD – Books on Demand, July 2020. ISBN: 978-1-78985-053-6.

[RL12] G. M. Ribeiro and G. Laporte. "An Adaptive Large Neighborhood Search Heuristic for the Cumulative Capacitated Vehicle Routing Problem". In: *Computers & operations research* 39.3 (2012), pp. 728–735.

[Rob14] C. Robert. *Machine Learning, a Probabilistic Perspective*. Taylor & Francis, 2014.

[RP06] S. Ropke and D. Pisinger. "An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows". In: *Transportation science* 40.4 (2006), pp. 455–472.

[RPH16] U. Ritzinger, J. Puchinger, and R. F. Hartl. "A Survey on Dynamic and Stochastic Vehicle Routing Problems". In: *International Journal of Production Research* 54.1 (Jan. 2016), pp. 215–231. ISSN: 0020-7543. DOI: 10.1080/00207543.2015.1043403.

[RVW06] F. Rossi, P. Van Beek, and T. Walsh. "Constraint Programming". In: *Handbook of constraint programming* (2006), pp. 1–31.

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[SC95] H. Simonis and T. Cornelissens. "Modelling Producer/Consumer Constraints". In: *International Conference on Principles and Practice of Constraint Programming*. 1995, pp. 449–462.

[Sch+] C. Schulte, G. Tack, M. Zayenz L., and Gecode devellopers. *GECODE*. `https://www.gecode.org/`.

[Sch+11a] P. Schaus, P. Van Hentenryck, J.-N. Monette, C. Coffrin, L. Michel, and Y. Deville. "Solving Steel Mill Slab Problems with Constraint-Based Techniques: CP, LNS, and CBLS". In: *Constraints* 16.2 (2011), pp. 125–147.

[Sch+11b] A. Schutt, T. Feydy, P. J. Stuckey, and M. G. Wallace. "Explaining the Cumulative Propagator". In: *Constraints* 16.3 (2011), pp. 250–282.

[Sch99] C. Schulte. "Comparing Trailing and Copying for Constraint Programming". In: *ICLP*. Vol. 99. 1999, pp. 275–289.

[SD+08] P. Schaus, Y. Deville, et al. "A Global Constraint for Bin-Packing with Precedences: Application to the Assembly Line Balancing Problem." In: *AAAI*. 2008.

[Sha98] P. Shaw. "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 1998, pp. 417–431.

[Sim+15] G. Simonin, C. Artigues, E. Hebrard, and P. Lopez. "Scheduling Scientific Experiments for Comet Exploration". In: *Constraints* 20.1 (2015), pp. 77–99.

[Sip12] M. Sipser. *Introduction to the Theory of Computation*. Cengage Learning, June 2012. ISBN: 978-1-285-40106-5.

[SKD95] A. Sprecher, R. Kolisch, and A. Drexl. "Semi-Active, Active, and Non-Delay Schedules for the Resource-Constrained Project Scheduling Problem". In: *European Journal of Operational Research* 80.1 (1995), pp. 94–102.

[Sli19] A. Slivkins. "Introduction to Multi-Armed Bandits". In: *Foundations and Trends® in Machine Learning* 12.1-2 (2019), pp. 1–286.

[SO08] H. Simonis and B. O'Sullivan. "Search Strategies for Rectangle Packing". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 2008, pp. 52–66.

[SO11] H. Simonis and B. O'Sullivan. "Almost Square Packing". In: *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer, 2011, pp. 196–209.

[Sol87] M. M. Solomon. "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints". In: *Operations research* 35.2 (1987), pp. 254–265.

[SPR19] D. Sacramento, D. Pisinger, and S. Ropke. "An Adaptive Large Neighborhood Search Metaheuristic for the Vehicle Routing Problem with Drones". In: *Transportation Research Part C: Emerging Technologies* 102 (2019), pp. 289–315.

[SS15] J.-J. Salazar-González and B. Santos-Hernández. "The Split-Demand One-Commodity Pickup-and-Delivery Travelling Salesman Problem". In: *Transportation Research Part B: Methodological* 75 (May 2015), pp. 58–73. ISSN: 0191-2615. DOI: 10.1016/j.trb.2015.02.014.

[Ste80] G. L. Steele Jr. *The Definition and Implementation of a Computer Programming Language Based on Constraints.* Tech. rep. Massachusets Inst. of Tech. Cambridge Artificial Intelligence Lab, 1980.

[Stu10] P. J. Stuckey. "Lazy Clause Generation: Combining the Power of SAT and CP (and MIP?) Solving". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Ed. by A. Lodi, M. Milano, and P. Toth. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 5–9. ISBN: 978-3-642-13520-0. DOI: 10.1007/978-3-642-13520-0_3.

[Sut63] I. E. Sutherland. "Sketchpad, a Man-Machine Graphical Communication System". PhD thesis. Massachusetts Institute of Technology, 1963.

[SW10] A. Schutt and A. Wolf. "A New ${\mathcal O}(N2\log n)$Not-First/Not-Last Pruning Algorithm for Cumulative Resource Constraints". In: *Principles and Practice of Constraint Programming – CP 2010*. Ed. by D. Cohen. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 445–459. ISBN: 978-3-642-15396-9. DOI: 10.1007/978-3-642-15396-9_36.

[SW11] R. Sedgewick and K. Wayne. "Algorithms: Intractability". In: *Algorithms (4th Edition)*. Addison-Wesley, 2011, pp. 910–921.

[SW12] D. Salvagnin and T. Walsh. "A Hybrid MIP/CP Approach for Multi-activity Shift Scheduling". In: *Principles and Practice of Constraint Programming*. Ed. by M. Milano. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 633–646. ISBN: 978-3-642-33558-7. DOI: 10.1007/978-3-642-33558-7_46.

[Tan+19] Y. Tan, M. Zhou, Y. Wang, X. Guo, and L. Qi. "A Hybrid MIP–CP Approach to Multistage Scheduling Problem in Continuous Casting and Hot-Rolling Processes". In: *IEEE Transactions on Automation Science and Engineering* 16.4 (Oct. 2019), pp. 1860–1869. ISSN: 1558-3783. DOI: 10.1109/TASE.2019.2894093.

[Tar78] R. E. Tarjan. "Complexity of Combinatorial Algorithms". In: *SIAM Review* 20.3 (1978). https://www.jstor.org/stable/2030351, pp. 457–491. ISSN: 0036-1445.

[Tho+] C. Thomas, Q. Cappart, P. Schaus, and L.-M. Rousseau. *CSPLib Problem 082: Patient Transportation Problem*. http://www.csplib.org/Problems/prob082.

[Tim02] C. Timpe. "Solving Planning and Scheduling Problems with Combined Integer and Constraint Programming". In: *OR Spectrum* 24.4 (Nov. 2002), pp. 431–448. ISSN: 1436-6304. DOI: 10.1007/s00291-002-0107-1.

[Tim17] Tim Stellmach. *Sudoku Puzzle*. https://commons.wikimedia.org/w/index.php?curid=57831926. Apr. 2017.

[Tod+17] R. Todosijević, A. Mjirda, M. Mladenović, S. Hanafi, and B. Gendron. "A General Variable Neighborhood Search Variants for the Travelling Salesman Problem with Draft Limits". In: *Optimization Letters* 11.6 (Aug. 2017), pp. 1047–1056. ISSN: 1862-4480. DOI: 10.1007/s11590-014-0788-9.

[TSH21] R. Turkeš, K. Sörensen, and L. M. Hvattum. "Meta-Analysis of Metaheuristics: Quantifying the Effect of Adaptiveness in Adaptive Large Neighborhood Search". In: *European Journal of Operational Research* 292.2 (2021), pp. 423–442.

[TV02a] P. Toth and D. Vigo. "1. An Overview of Vehicle Routing Problems". In: *The Vehicle Routing Problem*. Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Jan. 2002, pp. 1–26. ISBN: 978-0-89871-498-2. DOI: 10.1137/1.9780898718515.ch1.

[TV02b] P. Toth and D. Vigo. "Models, Relaxations and Exact Approaches for the Capacitated Vehicle Routing Problem". In: *Discrete Applied Mathematics* 123.1 (Nov. 2002), pp. 487–512. ISSN: 0166-218X. DOI: 10.1016/S0166-218X(01)00351-1.

[TV14] P. Toth and D. Vigo. *Vehicle Routing: Problems, Methods, and Applications*. SIAM, 2014.

[Uld+90] N. L. Ulder, E. H. Aarts, H.-J. Bandelt, P. J. Van Laarhoven, and E. Pesch. "Genetic Local Search Algorithms for the Traveling Salesman Problem". In: *International Conference on Parallel Problem Solving from Nature*. Springer, 1990, pp. 109–116.

[Van+16] S. Van Cauwelaert, C. Dejemeppe, J.-N. Monette, and P. Schaus. "Efficient Filtering for the Unary Resource with Family-Based Transition Times". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 2016, pp. 520–535.

[Vaz03] V. V. Vazirani. *Approximation Algorithms*. https://link.springer.com/book/10.1007/978-3-662-04565-7. 2003.

[VC88] P. Van Hentenryck and J.-P. Carillon. "Generality versus Specificity: An Experience with AI and OR Techniques". In: *National Conference on Artificial Intelligence*. 1988.

[VC99] P. Van Beek and X. Chen. "CPlan: A Constraint Programming Approach to Planning". In: *AAAI/IAAI*. 1999, pp. 585–590.

[VDS20] S. Van Cauwelaert, C. Dejemeppe, and P. Schaus. "An Efficient Filtering Algorithm for the Unary Resource Constraint with Transition Times and Optional Activities". In: *Journal of Scheduling* 23.4 (2020), pp. 431–449.

[Ver+20] H. Verhaeghe, S. Nijssen, G. Pesant, C.-G. Quimper, and P. Schaus. "Learning Optimal Decision Trees Using Constraint Programming". In: *Constraints* 25.3 (Dec. 2020), pp. 226–250. ISSN: 1572-9354. DOI: 10.1007/s10601-020-09312-3.

[Ver21] H. Verhaeghe. "The extensional constraint". https://dial.uclouvain.be/pr/boreal/object/boreal:252859. PhD thesis. UCL - Université Catholique de Louvain, 2021.

[VH15] P. Van Beek and H.-F. Hoffmann. "Machine Learning of Bayesian Networks Using Constraint Programming". In: *Principles and Practice of Constraint Programming*. Ed. by G. Pesant. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 429–445. ISBN: 978-3-319-23219-5. DOI: 10.1007/978-3-319-23219-5_31.

[Vil07] P. Vilím. "Global Constraints in Scheduling". PhD thesis. Charles University, 2007.

[Vil11] P. Vilím. "Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources". In: *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. 2011, pp. 230–245. DOI: 10.1007/978-3-642-21311-3_22.

[VLS15] P. Vilím, P. Laborie, and P. Shaw. "Failure-Directed Search for Constraint-Based Scheduling". In: *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. 2015, pp. 437–453. DOI: 10.1007/978-3-319-18008-3_30.

[VLS18] S. Van Cauwelaert, M. Lombardi, and P. Schaus. "How Efficient Is a Global Constraint in Practice?" In: *Constraints* 23.1 (2018), pp. 87–122.

[Wal02] T. Walsh. "Stochastic Constraint Programming". In: *ECAI*. Vol. 2. 2002, pp. 111–115.

[Wen+16] M. Wen, E. Linde, S. Ropke, P. Mirchandani, and A. Larsen. "An Adaptive Large Neighborhood Search Heuristic for the Electric Vehicle Scheduling Problem". In: *Computers & Operations Research* 76 (2016), pp. 73–83.

[Wol20] L. A. Wolsey. *Integer Programming*. John Wiley & Sons, 2020.

[XCSa] XCSP3 Team. *XCSP3*. http://www.xcsp.org/.

[XCSb] XCSP3 Team. *XCSP3 Instances*. http://www.xcsp.org/instances/.

[YH06] B. Yannou and G. Harmel. "Use of Constraint Programming for Design". In: *Advances in Design*. Ed. by H. A. ElMaraghy and W. H. ElMaraghy. Springer Series in Advanced Manufacturing. London: Springer, 2006, pp. 145–157. ISBN: 978-1-84628-210-2. DOI: 10.1007/1-84628-210-1_12.

[YH09] J. Yip and P. V. Hentenryck. "Evaluation of Length-Lex Set Variables". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 2009, pp. 817–832.

[YH11] J. Yip and P. V. Hentenryck. "Checking and Filtering Global Set Constraints". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 2011, pp. 819–833.

[YN97] T. Yamada and R. Nakano. "Job Shop Scheduling". In: *IEE control Engineering series* (1997), p. 134.

[YV10] J. Yip and P. Van Hentenryck. "Exponential Propagation for Set Variables". In: *Principles and Practice of Constraint Programming – CP 2010*. Ed. by D. Cohen. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 499–513. ISBN: 978-3-642-15396-9. DOI: 10.1007/978-3-642-15396-9_40.

[ZL10] R. Zamani and S. K. Lau. "Embedding Learning Capability in Lagrangean Relaxation: An Application to the Travelling Salesman Problem". In: *European Journal of Operational Research* 201.1 (Feb. 2010), pp. 82–88. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2009.02.008.