On the Art of Modelling for Constraint-Based Scheduling:

new methods, theory, and applications

A thesis submitted for the degree of

Doctor of Philosophy

by

Steven J. Edwards

Supervisors:
Prof. Andreas T. Ernst
Dr. Davaatseren Baatar
Prof. Kate Smith-Miles

School of Mathematics
Monash University, Australia

May, 2019

Copyright notice

Abstract

This thesis studies modelling choices in Constraint-Based Scheduling, which is the application of Constraint Programming (CP) to scheduling problems. CP is a problem-solving framework that establishes a clear distinction between (1) the *modelling* of an optimisation problem in terms of variables, constraints and an objective function, and (2) the algorithms that *solve* the model. Through the recent development of powerful automated solving procedures, some modern CP solvers are now able to model and solve broad classes of problems efficiently. This greatly reduces the challenge of solving many practical problems as end-users can now focus on understanding how best to model the problem they seek to solve. In fact, methods using off-the-shelf CP solvers now represent state-of-the-art techniques for many classical and industrial scheduling problems. Despite these advances, there still appears to be little theoretical insight into understanding why one model of a problem might in general outperform another.

The aim of this thesis is to understand how scheduling problems can be best modelled and solved by existing combinatorial solvers. In particular, the work is motivated by the scheduling of a fully-automated robotic system for the application of advanced cell staining: a process routinely used by pathologists to diagnose cancers and infectious diseases by enhancing cell visualisation. However, significant effort has been made to generalise insights as much as possible to allow for broader application.

The major contributions of the thesis include the following:

1. Symmetry Breaking - The thesis proposes methods for exploiting symmetry in models of multi-project scheduling problems with identical projects in order to accelerate solver performance. These methods are evaluated on both existing and modified models based on CP and Mixed-Integer Programming. The proposed symmetry breaking methods in general improve the performance of all the different approaches and in fact find a number of new best solutions on a well-established dataset.
2. Driving Variables and LU Consistency - The thesis formalises a method for developing models to CP problems where a feasible assignment over a subset of the variables can be efficiently extended to a complete solution. We name such subsets the driving variables of the problem, which extends the existing concept of backdoors to typical case complexity. The method is based on a novel local consistency measure that we name Lower Upper (LU) consistency.

We prove that a range of filtering algorithms and consistency checks from an existing CP solver enforce LU consistency under certain conditions and show how this concept can be used to significantly improve solver performance.

3. Interval Clusters - The thesis proposes a framework for modelling scheduling problems that reason over multiple levels of abstraction based on the novel concept of interval clusters. An interval cluster is a group of variables containing exactly one driving variable as well as satisfying a number of other conditions. To demonstrate the concept of interval clusters we model the real-world scheduling problem that motivates this work and demonstrate that the model can be solved efficiently.

The scheduling problems considered in this thesis range from the very abstract to a complex industrial scheduling problem. We initially provide an overview of relevant discrete optimisation techniques from the last few decades of research and discuss the strengths and limitations of alternative approaches. We then introduce a number of simplified problems that contain some real-world aspects and use these to test the effectiveness of the proposed ideas. The development of methods for solving these simplified problems are then generalised such that they can be used on the industrial scheduling problem we seek to solve. In this sense, the thesis achieves a balance between application-oriented research and fundamental research with more generalisable contributions.

Declaration

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature:.....

Steven J. Edwards, 30th June 2019

Acknowledgements

Firstly, thank you to my supervisors, Prof Andreas Ernst, Prof Kate Smith-Miles, and Dr Davaatseren Baatar for your guidance, encouragement, and support. I am extremely lucky to have been supervised by such a complementary group of people and appreciate our many discussions. I have learnt so much from working with you all.

There are many people from the School of Mathematics at Monash that I wish to acknowledge. Firstly, John and Gertrude, thank you for all of your support; we are incredibly lucky to have you both. To Chloé, your laughter is like sunshine and could light up even those dark times in research where nothing seemed to work. I am so grateful for your continuous support and friendship. To Simon and Andrew, thank you for the many brainstorms and for providing such a great place to ask such dumb questions. And to Carlos, Darcy, Michael, Joe, Cal, Anita, James, Kevin, Anurag, Dhananjay, Alin, Dana, Damien, Jens, Eduard, and the many other officemates, thank you for all of the memories.

I would also like to thank the wider research community. I have felt nothing but encouragement and support from the academics I have come into contact with during my candidature. In particular, I would like to thank Prof Mark Wallace for leading me down the path of Constraint Programming, and Dr Philippe Laborie for always providing very timely insights.

To my industry partner, Leica Biosystems, thank you for being an exciting company to work with and for providing such an interesting motivating problem. In particular, thank you to Matt Elvey-Price for being an excellent point of contact.

I am also grateful for being able to have spent time working from Canberra. Thank you to Dr Phil Kilby from Data 61, and Prof Tabrabata Ray from University of New South Wales, for making that possible. Also, an enormous thank you to Liz, Liv and Pepper at the Ritz; I could not have dreamt of a more idyllic place to live while writing up my thesis. I will cherish those times forever.

To my family - Mum and Dad, Tom and Emily (and Archie!), and MaMa and Pa - thank you for your ongoing love, support, and patience. And finally, to Anika, thank you for your constant belief in me, for riding the highs and lows with me, and for even proof-reading the thesis. You are a constant source of inspiration and I am so incredibly lucky to have you in my life.

Steven J. Edwards, June 2019

Contents

CHAPTER 1

Introduction

1.1 Motivation

This thesis is motivated by modelling and solving scheduling problems that arise when automating scientific experiments. More specifically, we are motivated by scheduling fully-automated robotic systems that complete cell staining processes in the fields of Immunohistochemistry (IHC) and In-Situ Hybridisations (ISH). For simplicity, we refer to these processes as *advanced cell staining*. Advanced cell staining considers the process of using chemicals and dyes to enhance the visualisation of cellular subcomponents of tissue samples under a microscope. The technique is commonly used to help pathologists identify the presence or prevalence of particular cell types, structures or microorganisms for the clinical diagnosis of cancers and infectious diseases. For an example of a tissue sample that has undergone cell staining, refer to Figure 1.1.

The ability to use dyes to better differentiate between cellular properties is a well established practice in the field of pathology. In fact, some staining techniques that were developed more than a century ago are still routinely used today, such as: (1) Haematoxylin and Eosin staining (H&E) (Mayer, 1891) used to detect the abnormalities in the organisation of cells typically associated with cancer, and (2) Giemsa staining (Giemsa, 1904) used to diagnose parasites such as malaria. Older still, the earliest reported usage of dyes to improve microscopic observations is

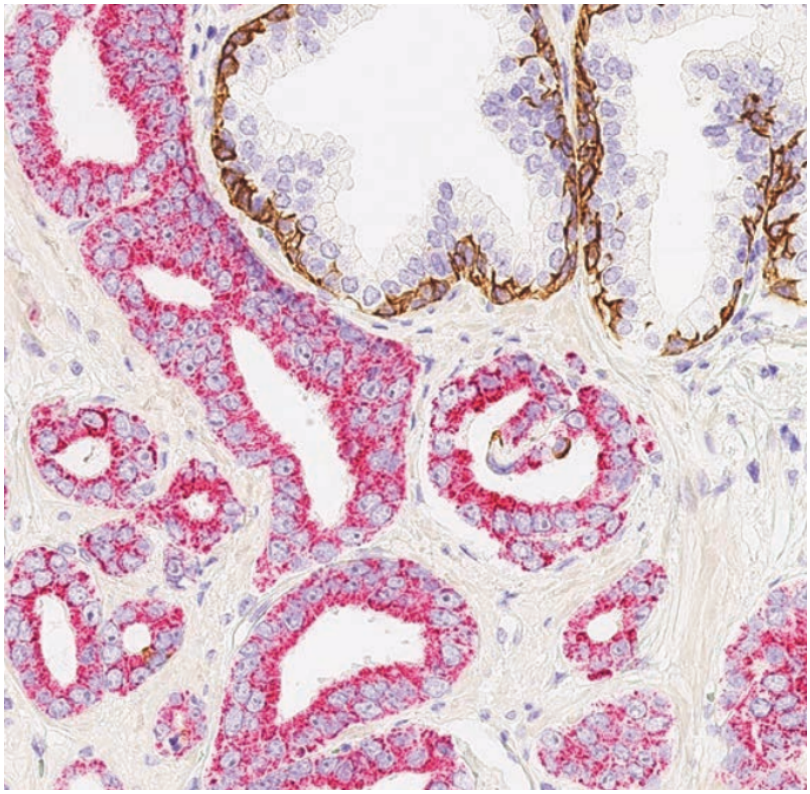


Figure 1.1: An example of a tissue sample after advanced cell staining. More specifically a user validated Prostate Marker Cocktail - HMW CK mouse antibody stains brown and p540S rabbit antibody stains red (Biosystems, 2012).

attributed to Hooke (1665). Given that significant examples have existed for well over a century which clearly demonstrate the practical importance of cell staining, it is unsurprising that the field has received extensive research since this time.

Today, advanced cell staining techniques allow pathologists to probe beyond pure gene morphology and test for the presence of specific proteins and RNA or DNA sequences. These techniques are all based on the sequential application of chemicals to a sample according to a very specific set of instructions commonly known as a staining protocol. Samples are generally sections of skin tissue taken from patients by biopsy. Although manual staining is still possible, in general advanced cell staining is typically completed by fully-automated systems such as that shown in Figure 1.2.

The need for automation is based on a number of factors. Firstly, due to the large number of steps in staining protocols, there is a risk of technical errors when performed manually, affecting variability in the quality, consistency and reliability of the results (Bánkfalvi et al., 2004). Secondly, as a single staining protocol can often take more than a couple of hours and multiple staining protocols are often

Figure 1.2: An image of Leica Biosystem's BOND-MAX Fully Automated IHC and ISH staining instrument (Biosystems, 2019).

used for a single sample from a single patient, automation increases the number of staining protocols a pathologist can complete in a given amount of time. The more tests a pathologist can complete, the more information they have available to reach an accurate diagnosis.

Automated systems vary in their design but in general can process dozens of different samples in parallel. We refer the interested reader to Prichard (2014) for a recent summary of the different automated instruments. Such systems generally consist of a range of different resources such as various types of liquid handling robots, heating/cooling apparatuses, and vacuums that must be coordinated in order to transfer chemicals to a set of samples such that they are processed according to their specified staining protocols.

This thesis is motivated by the scheduling of the resources of a system to process a large number of experiments efficiently while still adhering strictly to the specific staining protocols. In particular, we consider the next-generation advanced cell staining instrument developed by our industry partner Leica Biosystems, which has not yet been released to market.

While we are not aware of any existing literature specific to the scheduling of automated staining instruments, it is interesting to note that radiation therapy scheduling (Petrovic et al., 2013), i.e. the automated design and scheduling of radiation treatments, is a well-established optimisation problem in Operations Research.

Thus, although improved scheduling techniques are already been used to optimise cancer treatment, this thesis concentrates on the related process of optimising the diagnostic instruments themselves.

1.2 Research Aims and Methodology

The aim of this thesis is to provide theoretical insights and develop frameworks that can help understand how best to model and solve the types of scheduling problems that motivate this research. Although motivated by a very specific application, the aim of this thesis is not necessarily to develop a single application-specific algorithm. Our inclination to study more general techniques is based on the fact that the design of real-world systems can often change considerably during product development. By focussing on general solving approaches, this not only provided us with the flexibility to adapt to inevitable changes but also alleviated many of the issues surrounding the disclosure of sensitive information.

We restrict the scope of the thesis to understanding how the relevant problems are best modelled by existing combinatorial solvers such that they are solved efficiently. In doing so, we discovered that solvers based on Constraint Programming (CP) were particularly efficient. CP is a problem-solving framework that establishes a clear distinction between (1) the modelling of a discrete optimisation problem in terms of variables, constraints and objective function, and (2) the algorithms that solve the model. Much of the potential of CP has long been based on this distinction; as Freuder (1997) famously stated,

> *Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.*

Despite the goal of declarative problem solving, CP has not always adhered to a model-and-run paradigm (Puget, 2004). Not only do users have to specify how to model a problem in terms of variables, constraints and objective, but they also frequently have to specify how the search for a solution must proceed. In regards to Freuder's quote, the user must not only state the problem but also how the computer should solve it.

Recently, through the development of strong automated solving procedures, many CP solvers have aligned with the model-and-run paradigm. This was largely achieved in two distinct ways:

1. By complementing CP with the no-good learning technologies and strong

automated search developed for boolean satisfiability (SAT) problems such as by the solver Chuffed (Chu, 2011), and

2. By developing a scheduling specific modelling language on top of traditional CP and developing search processes specific to scheduling problems such as by IBM's CP Optimizer (Laborie et al., 2018)

As a result, CP methods using off-the-shelf solvers represent the state-of-the-art for many classical scheduling problems (Schutt et al., 2013c, 2011, 2013a; Vilim et al., 2015), as well as for many industrial scheduling problems (Booth et al., 2016; Kinable, 2015; Giles and van Hoeve, 2016; Kizilay et al., 2017; Kinnunen, 2016; Frank et al., 2016).

The benefits of having strong automated solving procedures is that it allows users to focus on how best to model the problem they wish to solve. Yet despite the recent advances of constraint-based scheduling techniques, there still appears to be little theoretical insight into understanding why one CP model of a problem might in general outperform another. Although the focus of the thesis is largely on solving technologies based on CP, where appropriate during the thesis, concepts are extended to Mixed Integer Programming (MIP) and relevant concepts from Project Scheduling. As mentioned, the work itself is particularly motivated by an industrial application, however significant effort has been made to generalise all insights as much as possible to allow for broader application.

1.3 Thesis Outline and Contributions

Each chapter of the thesis, excluding the introduction and conclusion, considers a single scheduling problem that relates to the motivating real-world problem. The relationship between these problems, which is visually summarised in Figure 1.3, uses two different approaches to simplifying the real-world problem that we refer to as (1) a top-down approach and (2) a bottom-up approach.

Before describing these approaches, we note the reasons why we consider special cases of the real-problem instead of describing and solving the motivating problem directly. Firstly, as previously stated the real-world problem is based on the development of an automated system by our industry partner that has not yet been released and thus is commercially sensitive. As such we wished to structure the research in such a way that any theoretical contributions could be shared with the research community independent of the immediate application. To that end, we restrict all of the potentially sensitive information to a single chapter of

the thesis, that is Chapter 6, which will not be made public until the product is launched. Secondly, our initial attempts to model and directly solve the problem in its full complexity were not immediately beneficial. We study restricted versions of the problem that offer insight into strategies to tackle the additional complexity. Thirdly, we aim to leverage and build upon existing methods in the scheduling literature and thus aimed to simplify the problem to connect it to classical problems.

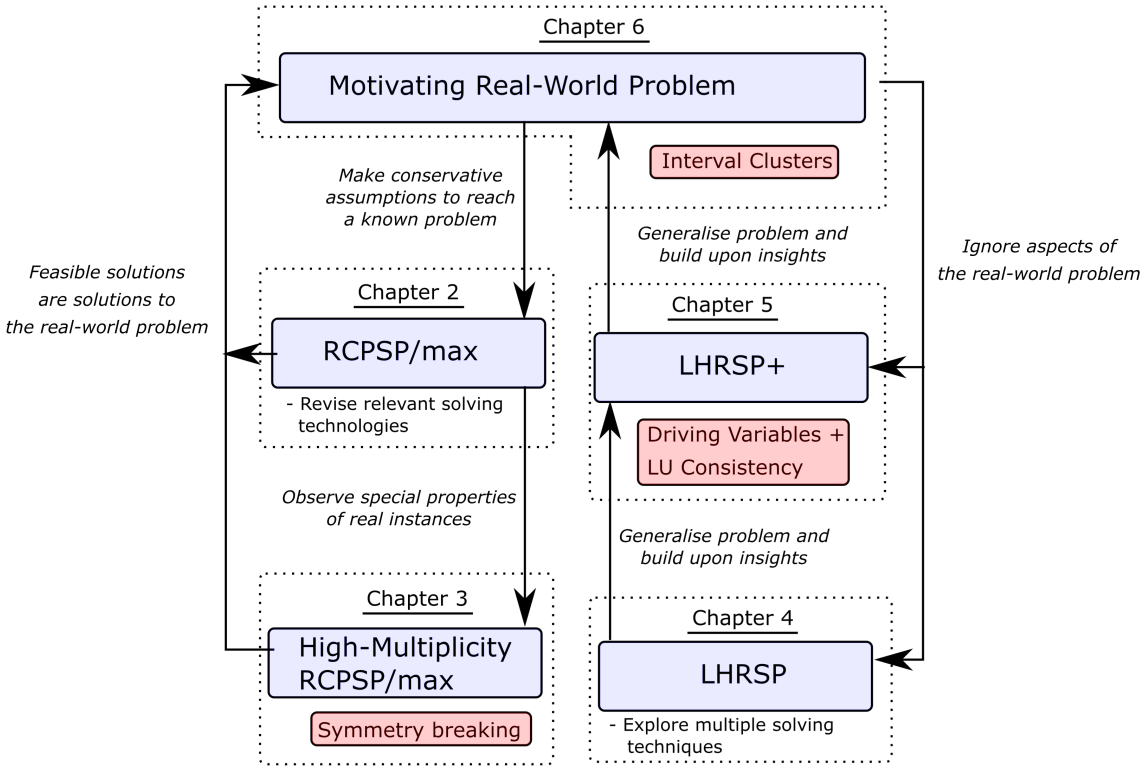


Figure 1.3: The thesis roadmap. The blue boxes indicate the main problems considered in the thesis. The red boxes indicate the main theoretical contributions of the thesis. The dashed boxes indicate where the problems and theories relate to the chapters of the thesis.

The top-down approach is based on making conservative simplifying assumptions to the real-world problem until we reach a classical problem in the literature. Examples of conservative assumptions include assuming a worst case travel time for the components that must move around the system, assuming that slides are placed into specific locations of the system, assuming that robots aspirate a single unit of chemical at a time, and assuming that tasks that the robots could do at the same time are done one at a time. Despite these assumptions being restrictive, there are multiple benefits of the top-down. Firstly, as all simplifying assumptions are conservative, any feasible solution obtained to the real-world instances of the simplified problem can be used directly as a feasible solution of the real-world

problem. Secondly, by mapping the real-world problem into a problem that exists in the literature, we can understand and leverage existing approaches. Thirdly, as was the case for us, when mapping real-world instances into an existing problem, these instances can have special properties that can be exploited. The main limitations of the top-down approach it does not necessarily provide optimal solutions to the real-world problem and it may not be immediately clear how to overcome the simplifying assumptions.

The bottom-up approach completely ignores some aspects of the real-world problem and starts with a sub-problem that is intuitive to state yet still challenging to solve. Once a satisfactory solution method has been developed for the simplified version of the problem for approximations of real-world instances, additional aspects of the real-world problem can be considered. This process is repeated until all remaining aspects are included. The limitation of the bottom-up approach is that solutions to the simplified versions of the problem cannot necessarily be immediately used as solutions to the real-world problem as some aspects are completely ignored. The benefit of the bottom-up approach is that: (1) it allows the development of insights into the structure of the problem that might not have been made on the full problem, (2) it can more easily compare a range of different approaches, and (3) it starts with and gradually considers problem-specific attributes of the real-world problem.

Clearly both the top-down and bottom-up approach have their respective benefits and limitations. In this thesis, Chapters 2 and 3 are based on the top-down approach, whereas Chapters 4 to 6 are based on a bottom-up approach. With this in mind, we will now provide an overview of the remaining chapters of the thesis.

Chapter 2: The aim of this chapter is to introduce the necessary terminology, concepts and solving technologies that are used throughout the thesis. This includes an introduction to relevant aspects of Project Scheduling, Mathematical Programming, Constraint Programming, and IBM's CP Optimizer. As a reference problem, this chapter considers the Resource-Constrained Project Scheduling Problem with Generalised Precedence Constraints (RCPSP/max), which is a well-studied generalisation of the classical Resource-Constrained Project Scheduling Problem (RCPSP). As shown in Figure 1.3, we consider the RCPSP/max because it is possible to map instances of the motivating problem into instances of the RCPSP/max by making conservative simplifying assumptions. It was observed that once mapped, these instances contained a lot of symmetry for which methods that exploit this observation were developed and presented in the following chapter.

Chapter 3: The aim of this chapter is to understand how symmetry can be exploited in instances of the RCPSP/max that include multiple projects with identical characteristics. We refer to these instances as high-multiplicity instances. These arise naturally in our application when the same staining protocol is used for multiple slides. The chapter is based on our journal article (Edwards et al., 2019), which is an extension of our conference paper (Edwards et al., 2017). We show that there exists symmetry between projects of the same class and propose two approaches of symmetry breaking: (1) adding additional constraints to the model in the form of precedence constraints, (2) remodelling the problem to reduce the number of variables. To test the usefulness of the symmetry breaking approaches a computational study is completed considering two families of discrete-time based MIP models and a number of state-of-the-art CP-based scheduling approaches. The study shows that both symmetry breaking approaches allow all solving methods to find and prove more optimal solutions. The best CP approach is then used to find a number of new best solutions to relevant problems from the MPSPlib, a multi-project scheduling problem library, whereas the best MIP approach is used to determine a number of tighter lower bounds.

Chapter 4: The aim of this chapter is twofold. Firstly, we aim to consider some challenging aspects of the motivating real-world problem that are not considered by the RCPSP/max. We do this by proposing and studying a problem that is simple and intuitive enough to be understood without needing to know the technical and potentially sensitive details, is challenging enough that developing good solving approaches is not immediately straightforward, and yet still has an interesting problem structure that can be exploited. Secondly, we wish to propose and compare a range of different approaches to how the problem can be modelled and solved. The chapter considerably extends the work of our workshop paper (Edwards et al., 2018). We introduce the Liquid Handling Robot Scheduling Problem (LHRSP) which seeks to minimise the time taken by a robot to transfer chemicals to a set of slides using a single pipette with finite capacity. We consider three different CP Optimizer models. The best approach exploits an observation that once a subset of the decision variables are fixed, and constraint propagation reaches a fixed point, the partial assignment can be extended immediately to a complete solution. This observation formed the basis for the next chapter.

Chapter 5: The aim of this chapter is to firstly formalise the insights made in the previous chapter of how solvers can be accelerated by only considering a subset of

the variables, and then demonstrate how this can be applied to solve a generalised version of the LHRSP, denoted LHRSP+, that contains some additional real-world aspects. More specifically, in many applications of CP it is often necessary to introduce auxiliary variables into a model either because it is difficult to express the constraints at all in terms of existing variables, or to improve propagation. An immediate consequence of additional variables however is that unless they are dealt with effectively they can significantly increase the search effort as there are more variables to choose from when branching. To help identify the subset of variables that should be used as choice points in the search tree, we introduce the notion of *driving variables*. Driving variables are a subset of the variables that once assigned values and constraint propagation reaches a fixed point, a search strategy can be determined in polynomial time that will always generate a complete assignment without backtracking. We distinguish our notion of driving variables from related concepts such as functional variable dependencies, backdoors to typical case complexity, and tractable structures for constraint satisfaction problems such as constraint networks. To ensure a search strategy will not backtrack we introduce a novel local consistency measure, Lower-Upper (LU) consistency, that can be used to reason about complete assignments. We then prove for a number of special cases of global constraints from CP Optimizer that existing filtering algorithms and consistency checks ensure LU consistency for various variable partitions.

Chapter 6: The aim of the chapter is to both describe the real-world motivating problem in detail and demonstrate how the theory of the previous chapters can be built upon to model and solve the problem. We introduce a framework for building models to Constraint-Based Scheduling systems that explicitly considers multiple levels of abstraction and can still be solved efficiently. More explicitly, we model the problem such that it is possible to partition the set of all variables into *interval clusters*, which represent subsets of variables containing exactly one driving variable as well as satisfying a number of sufficient conditions involving connectivity and boolean logic. In doing so, we structure the problem in such a way that each cluster can be thought of as a single higher level variable and the sufficient conditions ensure desirable behaviour over the remaining variables. We demonstrate the benefit of considering interval clusters when modelling and solving the next generation of fully automated advanced cell staining instruments.

Chapter 7: This chapter concludes the work done in this thesis and identifies opportunities for future research in the field.

CHAPTER 2

Background

This chapter introduces necessary terminology, concepts and solving technologies that are used throughout the thesis. The chapter is divided into four sections. The first section provides an overview of relevant scheduling concepts from Project Scheduling (PS). This includes describing the RCPSP/max, a well-studied generalisation of the classical RCPSP, which will be used as the reference problem throughout the chapter. The second section provides an overview of more than 50 years of trying to model and solve the RCPSP and its variants using MIP. Some limitations faced by MIP are discussed. The third section is an introduction to CP, in which an overview of fundamental aspects of the technology such as global constraints and search strategies are provided with a focus on scheduling problems. This sections also includes an introduction to *Lazy-Clause Generation*, a hybrid solving approach combining CP and boolean satisfiability problems (SAT). The fourth section provides a comprehensive introduction to CP Optimizer, which has been designed to model and solve industrial scheduling problems.

2.1 Project Scheduling Problems

PS considers a very broad class of optimisation problems that can be classified based on the attributes that define the problem. In general, these problems address the allocation of scarce resources to activities over time while trying to minimise some objective function. The field of PS is far reaching and overlaps other areas

of discrete optimisation such as Planning, Rostering and Routing — for example, it is well known that classical Vehicle Routing Problems (VRPs) can be viewed as variants of the Job Shop Problem (JSP), and vice versa (Beck et al., 2003). For the purpose of this thesis, we consider PS to be all problems that can be defined by the three-field classification scheme proposed by Brucker et al. (1999a), which importantly includes the RCPSP and its variants. A full review of the field of PS is outside the scope of this thesis; for a more complete reference the reader may refer to the recent handbooks by Schwindt and Zimmermann (2015a,b).

2.1.1 Classification Scheme

To structure the problems that fall under the umbrella of PS, many classification schemes have been proposed — the most famous of which is the classification scheme proposed by Brucker et al. (1999a). Classification schemes are extremely important as they allow the scientific community to keep track of which ideas are new, and which are already known. The classification scheme proposed by Brucker et al. (1999a) was inspired by those from machine scheduling, as machine scheduling is typically a special case of project scheduling problems. It is based on three fields: (1) the resource environment, (2) activity characteristics, and (3) the objective function. We briefly introduce a modified version of this scheme considering aspects relevant to the thesis.

Resource and Project Characteristics

- Single Machine (1): A single machine problem considers the case where there is a single renewable resource $|\mathcal{R}| = 1$, with a unit capacity, $R_0 = 1$. Renewable resources with unit capacity are also known as disjunctive resources as the activities requiring the resource cannot overlap in time.
- Project Scheduling with renewable resources (PS): These problems consider a set of renewable resource with a capacity that cannot be exceeded at any time in the time horizon.
- Project Scheduling with production consumption resources $(PS\pm)$: Production consumption resources (Neumann and Schwindt, 2003), also known as *reservoirs* (Laborie, 2003a), are resources where activities can either produce or consume units of resource at the start or end of the activity. The amount of resource available can be constrained to always be within a certain range.
- Multi-Project Scheduling with renewable resources (mPS): The multi-project

scheduling problem is a special case of project scheduling where multiple projects are considered. Precedence constraints are only defined between activities from the same project or from the master dummy start or end activities. Multi-project scheduling is discussed in more detail in Chapter 3.

- Job Shop (J): A job shop is a special case of multi-project scheduling with renewable resources. It considers a set of jobs, which can be thought of as projects for which the precedence constraints enforce a specific ordering on the activities in the project. Furthermore the problem considers a set of renewable resources with unit capacity, referred to as machines. The number of activities in each job is equal to the number of machines. Each activity in the job requires the use of a unique job for a certain amount of time.

Activity and Precedence Characteristics

Activity characteristics:

- Pre-emption ($prmp$): Pre-emption means that an activity can be interrupted even during processing and recovered later on. Pre-emptive activities can be separated into multiple activities of unit-time and ordered by simple precedence relations.
- Release dates ($release_i$): An activity / project can not be started before a certain time.
- Due dates (due_i): An activity / project must be completed before a certain time. Due dates can either be implemented as hard constraints that cannot be violated or soft constraints that incur a penalty cost to an objective function.
- Sequence-dependent setup times ($SDST$): Sequence-dependent setup times can be defined for pairs of activities that require the same disjunctive resources. Some simple practical examples of where sequence-dependent setup times occur are when a machine must change a tool before operating on a different type of activity, or when a machine must move from the location of the first activity to the location of the second.
- Activity Modes ($mode$): Activities can have multiple modes of execution, i.e., the problem not only decides *when* to complete the activity but, given an explicit set of options, *how* the activity is to be completed. Different modes of the same activity have different parameter information for characteristics such as duration and resource requirement.
- Mode-identity constraints ($ident$): Mode-identity constraints consider specific

modes from pairs or sets of different activities and enforce that all activities are completed in the associate mode or neither of them are (Rahimi et al., 2013; Drexl et al., 2000).

Precedence characteristics:

- Simple precedence constraints (*prec*): This parameter details for pairs of activities that the first activity must end before the second activity may start.
- Generalised precedence constraints (*temp*): This parameter details that generalised precedence constraints are to be considered, i.e., the precedences can specifies both a minimum and maximum waiting time between pairs of activities.
- Precedence Chains (*chain*): Assuming firstly that the problem only considers simple precedence relations, we say a problem has a precedence chain structure if each activity has at most one immediate predecessor and one immediate successor (Du et al., 1991). If generalised precedence constraints exist, an AoN network has a precedence chain structure if each activity has at most two neighbours; a predecessor with minimum and maximum waiting times and a successor with minimum and maximum waiting times. In general throughout this thesis we refer to projects with a precedence chain structure as *jobs*.

Objective Functions

There is a large range of different objective functions considered in the PS literature. Informally, objectives functions are considered *regular* if the objective function of a feasible schedule cannot be improved by only increasing the start times of (i.e., delaying) activities, and *irregular* otherwise. A more detailed discussion of shifts and schedule types is given in Section 2.1.2 with respect to Schedule Generation Schemes. A number of objective functions that are referred to throughout the thesis are as follows:

- Project Makespan (C_{max}): Minimise the time it takes to complete all of the activities considered, i.e., minimise the maximum completion time of any activity.
- Average Project Delay (APD): This objective is with respect to multi-project scheduling and aims to minimise the average that each project is delayed by. This can equivalently be interpreted as minimising the sum of the makespans from each individual project.

- Maximum Tardiness (L_{max}): The objective is to minimise the maximum lateness of any of the activities with respect to certain due dates.
- Net Present Value (NPV): In this formulation each activity has an associated cash flow which may be a payment (negative cash flow) or a receipt (positive cash flow). The cash flows are discounted with respect to certain discount rates, which makes it, in general, beneficial to execute activities with positive cash flows as early as possible, and negative cash flows as late as possible. Thus NPV is an example of an irregular objective function.

Solving Environment

Industrial scheduling problems have a number of practical challenges on top of the problems typically considered in the scheduling literature.

- *Static Scheduling*: *Static scheduling*, also known as *deterministic scheduling*, is based on the implicit assumption that the input data to the problem is precisely known in advance and no disruptions will occur when the schedule is implemented. The vast majority of scheduling problems considered in the literature are static scheduling problems. Static scheduling problems provide a great starting point to abstractions of real-world scheduling problems, where techniques can be developed independent to engineering challenges that arise during implementation.
- Dynamic Scheduling: *Dynamic scheduling* considers problems where the number of activities / jobs / projects can increase from the initial parameters provided. These problems are common in machine scheduling where jobs continuous arrive throughout the course of the day. The arrival of jobs are either assumed to be completely unpredictable or based on a stochastic process (Melchiors, 2015).
- Stochastic Scheduling: *Stochastic Scheduling* considers the situation where the input parameters are not precisely known and are thus modelled probabilistically, i.e., the durations of activities or resource requirements may be different to what is exactly specified (Melchiors, 2015).
- Reactive Scheduling: *Reactive scheduling* is based on revising or re-optimising an existing schedule when an unexpected event occurs (Herroelen and Leus, 2004) during the execution of the scheduling. Reactive scheduling is intimately related to both dynamic scheduling, as newly arrived activities must be incorporated into an existing schedule, and stochastic scheduling, as the initial parameters are discovered to be incorrect.

Problems can also be distinguished based on whether or not the time taken by the scheduling algorithm to determine a schedule impacts the problem itself.

- Offline Scheduling: *Offline scheduling* considers problems in which a schedule can be generated before it is executed. Hence in offline scheduling the execution time of the algorithm used to determine the schedule does not need to be taken into account
- Online Scheduling: *Online scheduling*, also known as *real-time scheduling*, considers the case where the time taken to determine the schedule must be taken into account by the schedule. Many scheduling problems in industrial settings are online scheduling problems.

The scheduling problems considered in this thesis are all static and offline. In reality though, the problems that motivated this thesis are dynamic, stochastic and online. The decision to narrow the focus of the thesis to static and offline versions of the problems was very deliberate. Many dynamic, stochastic, online scheduling problems can be solved by sequentially solving reactive scheduling problems. Zhao et al. (2013) propose such a framework for an industrial scheduling problem - the hoist scheduling problem. Hence we claim that any development made on the static and offline versions of the problem can be engineered to help the real-world systems.

2.1.2 Reference Problem - RCPSP/max

As claimed by Kolisch (2015), the RCPSP is probably the most studied problem in PS. The RCPSP have been extensively studied due to a combination of being easy to state, relevant in practice and yet hard to solve; being proven NP-Complete by Blazewicz et al. (1983). Some key scheduling concepts used in this thesis will be introduced with respect to a common generalisation of the RCPSP, that is the RCPSP/max. The problem description of the RCPSP/max is now given, a number of well-known related concepts stated, and then a working example is detailed that is returned to frequently throughout the chapter to solidify new concepts. With respect to the classification scheme the RCPSP/max is denoted $PS|temp|C_{max}$, whereas the RCPSP is denoted $PS|prec|C_{max}$.

Problem Description

An instance of the RCPSP/max considers a single project that consists of n activities has to be scheduled subject to generalised precedence relations and renewable resource constraints. The project is typically depicted as an Activity-on-Node (AoN) network with node set $V := \{0, 1, ..., n+1\}$ where each node represents an activity and nodes 0 and $n+1$ are dummy activities representing the start and end of the project, respectively. A generalised precedence relation between two activities $i, j \in V : i \neq j$, is represented by an arc (i, j) with arc weight $\delta_{i,j}$. Let A denote the set of all arcs of a project and $\mathcal{R}$ denote the set of all the renewable resources required to undertake the project. Each renewable resource $k \in \mathcal{R}$ has a capacity of R_k. In order to be processed activity $i \in V$ requires $r_{i,k}$ units of renewable resource $k \in \mathcal{R}$ for every period of its duration d_i without interruption, i.e., *pre-emption* is not allowed.

A schedule $S = (S_0, S_1, ..., S_n + 1)$ defines for each activity $i \in V$ the start time, S_i. A schedule is said to be *time-feasible* if for each arc $(i, j) \in A$, the following is true

$$S_j - S_i \geq \delta_{i,j}.$$

A schedule is said to be *resource-feasible* if at each time period $t \in H$ across the time-horizon, H, the demand for each resource $k \in R$ does not exceed its capacity R_k, i.e.,

$$r_k(S, t) = \sum_{\substack{i \in V: \\ S_i \leq t < S_i + d_i}} r_{i,k} \leq R_k.$$

A schedule is called *feasible* if it is both time- and resource-feasible. The objective is to determine a feasible schedule such that the time required for performing all activities, i.e., S_{n+1}, is minimised. This objective function is known as minimising the *makespan* of the project.

Relevant Theory

The generalised precedence relations as defined above are represented in so-called *standardized form*, i.e., they are defined between the start times of the two activities. Bartusch et al. (1988) make the observation that as pre-emption is not allowed for activities the end-start, start-end, and end-end relations can be transformed to start-start relations by the so-called Bartusch et al transformations. If pre-emption is allowed, or the duration of the activities is a variable in the problem, the end-start, start-end, and end-end precedences must be considered explicitly.

Generalised precedence relations can describe both *minimum* and *maximum time lags*. Consider the case where between two activities, there is both a minimum and maximum amount of waiting time between the start of the two activities, i.e., for $i, j \in V$, $\delta_{i,j}^{min} \leq S_j - S_i \leq \delta_{i,j}^{max}$. The minimum time, $\delta_{i,j}^{min}$, is called the minimum time lag, and the maximum time, $\delta_{i,j}^{max}$, is called the maximum time lag. To model the minimum time lag as a generalised precedence constraint, an arc (i, j) is introduced into the AoN network with arc weight $\delta_{i,j} := d_{i,j}^{min}$. To model the maximum time lag as a generalised precedence constraint, an arc (j, i) is introduced with $\delta_{j,i} := -\delta_{i,j}^{max}$. This is a valid operation as $S_i - S_j \geq -\delta_{i,j}^{max} \iff S_j - S_i \leq \delta_{i,j}^{max}$. Throughout this thesis we frequently refer to generalised precedence constraints with non-negative arc weights as minimum time lags, and those with negative arc weights as maximum time lags.

The classical RCPSP is a special case of the RCPSP/max where for each arc $(i, j) \in A$ the weight of the arc is equal to the duration of the predecessor activity, i.e., $\delta_{i,j} = d_i$. The precedences in the RCPSP are more commonly expressed as end-start precedences with arc-weight 0 and commonly referred to as *simple precedences*. As all activities are defined to have a non-negative duration, there do not exist any maximum time lags. As the RCPSP is a special case of the RCPSP/max and it is known that the RCPSP is strongly NP-Hard, then so too is the RCPSP/max. Furthermore the problem of detecting whether a feasible solution exists for the RCPSP/max is NP-complete (Bartusch et al., 1988).

Working Example

The following example will be used frequently throughout the chapter to exemplify a number of concepts.

Example 2.1.1. Consider an instance of the RCPSP/max with a single renewable resource and five activities. The single resource has a capacity of 4. The five activities are labelled A, B, C, D, and E, and have durations of 2, 8, 3, 1, 4 respectively, and resource requirements of 3, 2, 1, 2, 2, respectively. There are five generalised precedence constraints: $S_A + 2 \leq S_B$, $S_B + 4 \leq S_C$, $S_C - 9 \leq S_A$, $S_D + 4 \leq S_E$, and $S_E - 4 \leq S_D$. The AoN network associated with this instance is given in Figure 2.1. An optimal solution of this instance is represented as a Gantt chart in Figure 2.2.

Schedule Generation Schemes

Schedule generation schemes are the backbone of heuristics to solve project scheduling problems (Kolisch, 2015). Here we first introduce the different types of sched-

Figure 2.1: The activity-on-node network of the example problem

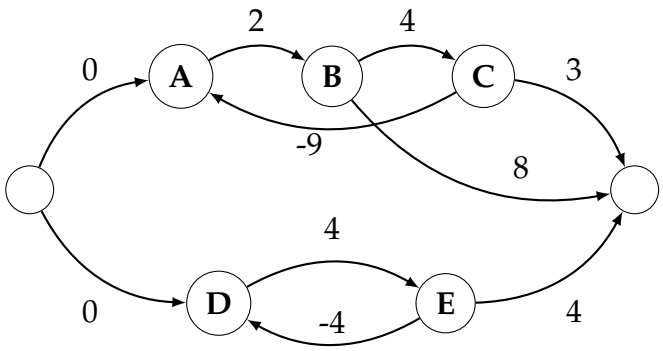
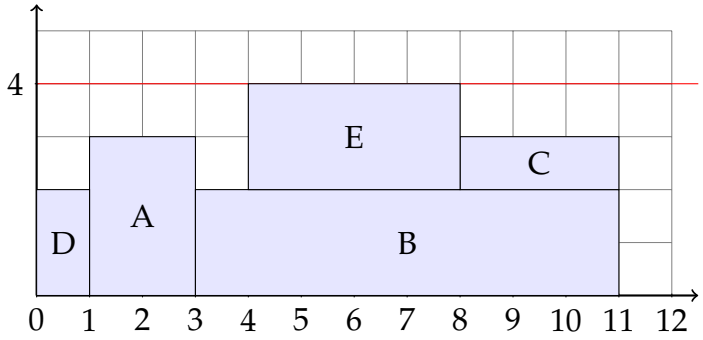


Figure 2.2: An optimal solution to the example problem



ules, then state the serial and parallel schedule generations with respect to the RCPSP, and then discuss the challenges of generalising this to the RCPSP/max.

Active, Semi-Active, Non-Delay Schedules With respect to the RCPSP, Sprecher et al. (1995) introduced the notion of *active*, *semi-active*, and *non-delay* schedules. These different schedule types formed the basis of analysis for comparing special types of heuristics known as *schedule generation schemes*. First we must introduce the notion of a *shift*. A shift is of an activity $i \in V$ with respect to a feasible schedule S is an operation which derives from S another feasible schedule S' such that activity i is changed and all other activities remain the same, i.e., $S'_i \neq S_i$ and $S'_j = S_j$ for $j \in V \setminus \{i\}$. A *left-shift* of activity $i \in V$ is a shift such that activity i starts earlier in the updated schedule, i.e., $S'_i < S_i$ and $S'_j = S_j$ for $j \in V \setminus \{i\}$. A *one-period left shift* of activity $i \in V$ is a left shift such that $S_i - S'_i = 1$. A *local left shift* of an activity i is a left shift of activity i which is obtained by one or more successively applied one-period left shifts of activity i. This implies that each intermediate schedule derived from a local left shift is also feasible. A *global left shift* of an activity i is a left shift of activity i which is not obtainable by a local left shift. Hence for a global left shift $S_i - S'_i > 1$ and at least one of the intermediate schedules in the course

of the global left shift is not feasible. Equivalent definitions can be made for right shifts.

The notion of shifts is also used to classify different objective functions. An objective function is said to be *regular* if a solution cannot be improved through performing right shift operations. Objective functions that are not regular are *irregular*. Minimising makespan is clearly an example of a regular objective function as it is not possible to finish the project earlier by starting activities later than they currently are.

The schedule types are now defined as follows. An *active schedule* is a feasible schedule where none of the activities can be locally or globally left shifted. A *semi-active schedule* is a feasible schedule where none of the activities can be locally left shifted. A *non-delay* schedule is an active schedule even if each activity $i \in V$ with duration d_i is split into a series of d_j activities with duration 1 and simple precedences are between consecutive sections, i.e., the first time unit must finish before the start of the second time unit and so on. Sprecher et al. (1995) then prove that the set of feasible schedules is a subset of all schedules, the set of semi-active schedules is a subset of all feasible schedules, the set of all active schedules is a subset of all semi-active schedules, and the subset of all non-delay schedules is a subset of all active schedules. The set of schedules with the smallest cardinality that contains an optimal solution with respect to a regular objective function is the active set, i.e., the set of non-delay schedules does not necessarily contain an optimal schedule.

Serial Schedule Generation Scheme The serial schedule generation scheme (SSGS) inserts activities into a schedule one-by-one at their earliest feasible start time. For a scheduled activity there are no local or global left shifts possible. Hence SSGSs generate active schedules (Kolisch, 1996). First let $\mathcal{C}_\mu$ be the set of activities which have been scheduled up to iteration μ, known as the *completed set*. Let $\mathcal{A}(\mathcal{C}_\mu, t) = \{i \in \mathcal{C}_\mu | S_i \leq t < S_i + d_i\}$ be the set of activities in the completed set in progress at time $t \in H$, known as the *active set*. Now, let $\tilde{R}_k(t) = R_k - \sum_{i \in \mathcal{A}(\mathcal{C}_\mu, t)} r_{i,k}$ be the remaining capacity of resource k at time t. Finally let $\mathcal{D}_\mu$ be the *decision set* containing the activities that have not yet been scheduled but all of their predecessors have been scheduled, i.e., $\mathcal{D}_\mu = \{i \in V \setminus \mathcal{C}_\mu | Pred(i) \subseteq \mathcal{C}_\mu\}$.

The SSGS takes n iterations. In line 1, the dummy start node is set to start at 0, is inserted into the completed set and the resource profile is empty. Then for each iteration the following steps are completed. Firstly the decision set and the

Algorithm 1 Serial Schedule Generation Scheme (SSGS) for the RCPSP

1: Initialization $S_0 = 0, \mathcal{C}_0 := \{0\}, \tilde{R}_k(t) = 0$
2: **for** each job μ **do**
3: Update $\mathcal{D}_\mu$ and $\tilde{R}_k(t)$ for all $k \in \mathcal{R}$ and $t \in H$
4: Select $i \in \mathcal{D}_\mu$
5: $EST_i := \max_{j \in Pred(i)}(S_j + d_j)$
6: $S_i := \min\{t | EST_i \leq t, r_{i,k} \leq \tilde{R}_k(\tau)$ for all $k \in \mathcal{R}$ and $\tau = t, ..., t + d_i - 1\}$
7: $\mathcal{C}_\mu := \mathcal{C}_{\mu-1} \cup \{i\}$

resource profile are updated, line 3. Then one of the activities from the decision set is selected, line 4, and its earliest start time in calculated, line 5. From the earliest start time, the activity is inserted into the first slot that does not violate the resource capacities of any of the resources. Finally the activity is added to the completed set.

Kolisch and Hartmann (1999) observe that when searching for the earliest resource feasible start time S_i in line 6 not every t has to be checked. Instead, the resource feasibility only needs to be checked at the end time of activities that have already been completed, thereby reducing the time complexity from the naive implementation of $\mathcal{O}(n \cdot T \cdot |\mathcal{R}|)$ to $\mathcal{O}(n^2 \cdot |\mathcal{R}|)$.

Priority Rules In order to decide which activity should be chosen from the decision set in line 4, there are two differing frameworks: *priority rules* and *activity lists*. Priority rules are, as the name implies, rules for determining the priority of the activities to be considered in the decision set. For example, a well-known priority rule is the *minimum slack priority rule*, where the slack of an activity $i \in V$ is the difference between its latest and earliest start times, $LST_i - EST_i + 1$. When the minimum slack priority rule is being utilised, the activity with the minimum slack compared with other activities in the decision set is selected. Ties can either be broken randomly or based on the index of the activities.

Much research has been undertaken to determine which priority rules for the RCPSP are the best. Klein (2000) benchmarked 73 different priority rules for the RCPSP and evaluated them experimentally. Clearly the best choice of priority rule will depend on the problem instance. More recently, genetic programming has been used to automate the process of determining good quality priority rules (Chand, 2018; Chand et al., 2018).

List heuristic An activity list, $\ell := \{i_1, i_2, ..., i_n\}$, is determined before the start of the algorithm and used to determine which activity in the decision set to se-

lect. The activity list is *precedence feasible* if for each activity i_k in the list, each immediate predecessor in the AoN, $Pred(i_k)$, has to be before i_k in the list, i.e., $Pred(i_k) \subseteq \{i_1, ..., i_{k-1}\}$ holds. The SSGS applied to activity lists is employed by many metaheuristics for the RCPSP, as summarised by Kolisch and Hartmann (2006).

Parallel Schedule Generation Scheme The parallel schedule generation scheme is time oriented, whereby activities are inserted into the schedule in a monotonically increasing order with respect to start time. Each iteration μ has an associated start time t_μ. The active, decision and completed sets must now be redefined with respect to t_μ. The completed set $\mathcal{C}'_\mu$ consists of all the activities such that the finish time is less than or equal to t_μ, i.e., $\mathcal{C}'_\mu := \{i \in V | S_i + d_i \leq t_\mu\}$. The active set, $\mathcal{A}'_\mu$, consists of all the activities that are currently in process at t_μ, i.e., $\mathcal{A}'_\mu := \{i \in V | S_i \leq t_\mu < S_i + d_i\}$. The remaining capacity of resource $k \in \mathcal{R}$ at time t_μ is $\tilde{R}'(t_\mu) = R_k - \sum_{i \in \mathcal{A}'_\mu} r_{i,k}$. Finally the decision set is now defined as the set of all activities that are not yet completed, all the predecessors of which have been completed and at time t_μ there is enough remaining capacity in order to be processed, i.e., $\mathcal{D}'_\mu := \{i \in V \setminus \{\mathcal{C}'_\mu \cup \mathcal{A}'_\mu\} | Pred(i) \subseteq \mathcal{C}'_\mu$ and $r_{i,k} \leq \tilde{R}'_k(t_\mu)$ for all $k \in \mathcal{R}\}$. Kolisch and Hartmann (1999) state that the time complexity of the PSGS is also $\mathcal{O}(n^2 \cdot |\mathcal{R}|)$.

Algorithm 2 Parallel Schedule Generation Scheme (PSGS) for the RCPSP

1: Initialization $\mu := 0$, $t_\mu := 0$, $\mathcal{C}'_0 := \emptyset$, $S_0 = 0$, $\mathcal{A}'_0 := \{0\}$
2: **while** $|\mathcal{C}'_\mu \cup \mathcal{A}'_\mu| \leq n$ **do**
3: $\quad \mu := \mu + 1$
4: $\quad t_\mu := \min_{i \in A'_{\mu-1}}(S_i + d_i)$
5: $\quad$ Update $\mathcal{C}'_\mu, \mathcal{A}'_\mu, \tilde{R}'_k(t_\mu)$ for all $k \in \mathcal{R}, \mathcal{D}_\mu$
6: $\quad$ **while** $\mathcal{D}'_\mu \neq \emptyset$ **do**
7: $\quad\quad$ Select $i \in D'_\mu$
8: $\quad\quad S_i := t_\mu$
9: $\quad\quad$ Update $\tilde{R}'_k(t_\mu)$ for all $k \in \mathcal{R}$, as well as $\mathcal{A}'_\mu, \mathcal{D}'_\mu$.
10: $S_{n+1} = max_{i \in Pred(n+1)} S_i + d_i$

The algorithm first sets the dummy start node to zero and assigns the dummy start to the active set, line 1. Each iteration has a unique counter μ and a unique schedule time t_μ, set in lines 3 and 4 respectively. The unique schedule time is the earliest end time of the activities in process in iteration $\mu - 1$. For the new iteration, and schedule time, the sets of completed and active activities, the remaining capac-

ity of the resources, and the decision set are calculated in line 5. Following this, the start times of activities from the decision set are iteratively assigned to μ until the decision set is empty, in lines 7- 9. Once all of the n activites have been scheduled, the start time of the dummy end activity $n+1$ is fixed in line 10.

Priority rules and activity lists can again be used when deciding which activity to select from the decision set, in this case in line 7. In the case where an activity list is used, the activity from the decision set is chosen which is on the foremost position of the list.

A schedule constructed by the PSGS belongs to the set of non-delay schedules (Kolisch, 1996). Therefore the PSGS is not necessarily capable at generating the optimal solution. However in practice as the cardinality of the set of non-delay schedules is smaller than the set of active schedules, in general fewer schedules must be generated when searching for a good schedule with the PSGS compared with the SSGS. This was experimentally verified by Kolisch and Sprecher (1997), who observed that for a simple priority based heuristic, independent of the specific priority rule employed, the average objective function value when employing the PSGS outperforms those obtained with the SSGS.

Schedule Generation Schemes for RCPSP/max

Much research in the mid to late 1990s focussed on generalising the schedule generation schemes generated for the RCPSP to the more challenging RCPSP/max. A consequence of the maximum time lags is that it is no longer possible to continuously right-shift an activity to find a feasible starting time in the schedule. Consequently, it is no longer possible to safely insert activities into the schedule one at a time without encountering the possibility of infeasibility. An overview of the priority rules methods of Zhan (1994), Neumann and Zhan (1995), and Brinkmann and Neumann (1996) can be found in Brucker et al. (1999a) and Neumann and Zimmermann (1999). In general all of these methods introduced the notion of an *unscheduling step*.

The unscheduling step, as reviewed by Franck et al. (2001), is performed as follows. In Line 6 of the SSGS, the selected activity, $i \in V$, is set to the earliest starting time that does not violate the resource capacities, let t^* denote this time. However when maximum time lags are considered it is now possible for this scheduling time to be greater than the latest starting time of the activity, $t^* > LST_i$. This can only be due to a maximum time lag between i and one or more activities in the completed set. Therefore a subset of the activities in the completed set must now

be unscheduled. Let this *unschedule set* be defined as $\mathcal{U} := \{j \in C | LST_i = S_j - \delta_{i,j}\}$. All activities $j \in \mathcal{U}$ are unscheduled and their earliest start times are increased by $t^* - LST_i$. As the number of unscheduling steps may be exponential with respect to the number of activities, typically a maximum number of unscheduling steps is prescribed, after which the algorithm terminates without a solution. The algorithm then continues as usual.

The connectivity of the AoN network can be used to help schedule the activities to reduce the number of unscheduling steps required. Recall that a *strongly connected component* of a directed graph is a set of at least two nodes such that every vertex in the set is reachable from every other vertex following the directions of the arcs. Neumann and Zhan (1995) observe that a project network has a feasible schedule if and only if for each strongly connected component of the AoN network, there is a feasible subschedule. Moreover, if all of the arcs in all of the strongly connected components are contracted, i.e., strongly connected components are represented by a single node, the resulting graph, referred to as the *contracted graph*, contains no cycles, and thus no negative time lags. The remaining precedences in the contracted graph thus enforce the order in which the strongly connected components can be considered for scheduling. Once all the activities from the strongly connected component that corresponds to the root node in the contracted graph have been scheduled, then assuming that all strongly connected components have a feasible subschedule, a feasible schedule exists that respects this subschedule. Heuristics such as the direct method introduced by Franck et al. (2001), take advantage of this property. Hence once all activities from a strongly connected component have been successfully incorporated into the schedule, these activities will never have to be unscheduled.

Example 2.1.2. Considering the instance of the RCPSP/max from Example 2.1.1 there are two strongly connected components; (1) nodes A, B, and C, and (2) nodes D and E. Furthermore, if considered in isolation, there clearly exist feasible subschedules for each of the subcomponents, e.g., $S_A = 0, S_B = 2, S_C = 6$ for the first component and $S_D = 0, S_E = 4$ for the second. If all the edges within both components are contracted, there does not exist a precedence relation between the two components.

In the following example we see how the unscheduling step uses the maximum time lags to learn values that would also lead to a failure.

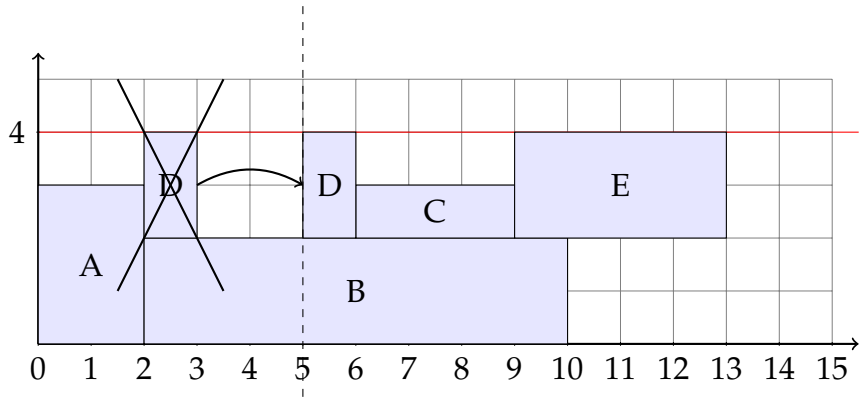
Example 2.1.3. Consider scheduling instance of the RCPSP/max from Example 2.1.1 with a list scheduling algorithm with list $\mathcal{L} = \{A, B, C, D, E\}$. The algorithm

begins by selecting activity A and assigning it to its lower bound, $S_A = 0$. The earliest start of activity B is updated due the minimum time lag between A and B, $EST_B = 2$. Likewise the earliest start time of C is updated due to the precedence with B, $EST_C = 6$. Activity B is then assigned to its earliest start time, $S_B = 2$, as that does not violate the resource capacity. Activity C is assigned to its earliest start time, $S_C = 6$. From Example 2.1.2, we know that as A, B, and C constitute a strongly connected component in the AoN network (without any proceeding components), there must exist a feasible schedule to the instance with this subschedule.

The algorithm proceeds to assign activity D a start time of 2 as that is the earliest time after its current earliest start time that would be resource feasible. The earliest start time of E is updated to be 6 due to the minimum time lag with D. Activity E is then assigned to start at 9 as that is the first time after its earliest start time that would be resource feasible. However this start time violates the maximum time lag between D and E that ensures that $S_D + 4 \geq S_E$. The earliest start time of D is then set to 5 as that would be the earliest time that would not violate the maximum time lag, then activities D and E are removed from the schedule. The algorithm then continues to assign D a start time of 5 and E to 9, which is now feasible. As a schedule has been found the algorithm terminates.

The insight of the unscheduling step is that the algorithm never tries to assign activity D to start times of 3 and 4 as they are known to lead to infeasible schedules.

Figure 2.3: The unscheduling step in schedule generation schemes of the RCPSP/max is used to skip over sections of the search that are known to violate a maximum time lag. The graph below illustrates a section of Example 2.1.3 where the earliest start time of activity D is moved from 2 to 5 so that the maximum time lag with E is not violated.



2.2 Mixed Integer Programming

Mixed Integer Programming (MIP) considers problems that can be modelled as a set of n variables, m linear constraints and a linear objective function such as,

$$\min \quad \sum_{i=1}^{n} c_i x_i \tag{2.1a}$$

$$s.t. \quad \sum_{i=1}^{n} a_{i,j} x_i \leq b_j \quad \forall j \in [1, ..., m] \tag{2.1b}$$

$$x_i \geq 0 \quad \forall i \in [1, ..., n] \tag{2.1c}$$

$$x_i \in integer \quad (\text{for some or all of } i \in [1, ..., n]) \tag{2.1d}$$

where $a_{i,j}$, b_j, and c_i are parameters of the problem. An overview to MIP is significantly outside the scope of this thesis. Due to the success of MIP in many fields of discrete optimisation, many different models have been proposed applying MIP to PS, particularly to the RCPSP. In fact, the first MIP models for the RCPSP were proposed over 50 years ago in the seminal work of Pritsker and Watters (1968). An excellent recent summary of the different approaches is given by Artigues et al. (2015), however this does not account for the recent work on compact models completed by Tesch (2016, 2018a). Significantly more effort has been placed on applying MIP to the RCPSP, compared with the RCPSP/max. However in general these models can be trivially extended to account for generalised precedence constraints and hence for now we revise the models for the RCPSP.

2.2.1 Models

Here we briefly revise the known models to the RCPSP in order of decreasing number of variables. The number of variables is typically a function of the number of activities n as well as the time-horizon T considered. Note that in general the time horizon is significantly larger than the number of activities, $T > n$. The number of precedence constraints theoretically can be quadratic with respect to the number of activities. However in the majority of practical applications, including those that motivate this thesis, the number of precedences is linear with respect to the number of activities.

$\mathcal{O}(T^n)$:

- *Chain-Decomposition formulation:* The model is based on the decomposition of the precedence constraints of the AoN network into chains. A binary variable is introduced for each chain and each possible subschedule that the chain can feasibly have. Considering the case where all activities belong to a single chain, and for simplicity all activities have zero duration, there are $\binom{T}{n}$ possible feasible subschedules, which corresponds to an order of $\mathcal{O}(T^n)$ variables required. Furthermore as $\mathcal{O}(T^n) \geq \mathcal{O}(n^n) \geq \mathcal{O}(n!)$ the model is worse than factorial with respect to the number of activities — making it clearly impractical to ever implement directly. The model was originally proposed by Kimms (2010) as a column-generation procedure to compute upper bounds of the net present value maximization problem.

$\mathcal{O}(T \cdot 2^n)$:

- *Feasible-Subset formulation:* A feasible subset is a set of activities that can be in progress simultaneously without exceeding the resource availability of any renewable resource and are not pairwise linked by a simple precedence constraint. Considering the trivial case where all activities in the project can form a feasible subset, the number of feasible subsets are $\sum_{k=0}^{n} \binom{n}{k} = 2^n$. A MIP model based on the concept of feasible subsets was initially proposed by Mingozzi et al. (1998). A binary variable is introduced for each time point and each feasible subset representing the case where all activities in that subset are in progress at that time, hence requiring $\mathcal{O}(T \cdot 2^n)$ possible binary variables.

$\mathcal{O}(T \cdot n)$:

- *Discrete-time formulations:* Discrete-time formulations have been widely studied for project scheduling problems. This is due to their relatively strong LP relaxations and to their ability for being extended to incorporate different constraints and objective functions. These models are based on decision variables of the type $x_{i,t}$ indicating a particular status (start, in-progress or complete) of activity $i \in V$ at time $t \in H$. As the number of variables is linear in function of the scheduling horizon, the standard variants of these models are pseudo-polynomial size with $\mathcal{O}(T \cdot n)$ binary variables. As a *compact* MIP formulation is defined to have a polynomial number of variables and constraints, the discrete-time formulations are not compact formulations.
  - *Variable Definitions:* It is worth mentioning at this point that discrete-time formulations can be classified into three groups based on what the binary variables represent. Firstly, a *pulse-start* variable $x_{i,t}$ equals

1 if activity $i \in V$ starts at exactly $t \in H$, and 0 others. Models based on pulse-start variables were proposed by Pritsker and Watters (1968) Pritsker et al. (1969). Secondly, a *step* variable $x_{i,t}$ equals 1 if activity $i \in V$ starts at time $t \in H$ or before, and 0 otherwise. Although presented as new by Klein (2000), models containing step variables were also proposed by Pritsker and Watters (1968). Thirdly, an *on/off* variable $x_{i,t}$ equals 1 if activity $i \in V$ is in progress at time $t \in H$, and 0 otherwise. On/Off variables were first motivated by the RCPSP where pre-emption was allowed (Kaplan, 1998).

It is well known that it is possible to obtain formulations based on the different variables from one another through non-singular transformations of the binary variables, and these different formulations yield the same LP relaxation (Artigues, 2017). Moreover, Kopanos et al. (2014) showed experimentally that in practice solvers can benefit from models that have combinations of the different variable types. These benefits are a result of formulating models that modern MIP solvers can take advantage of. For example, formulations based on step-variables have a relatively sparser constraint matrix, which modern solvers can take advantage of, yet this comes at the compromise that the formulations of the constraints makes it more difficult for the solvers to automatically infer cuts during search. Hence these variables can be augmented with, e.g. on/off variables, such that the constraint matrix is still sparse and cuts can easily be inferred by the solvers.

– *Variations:* There also exist variations of discrete-time models where the time discretisation is non-uniform, also known at *time-buckets* (Hurink et al., 2002).

$\mathcal{O}(n^3)$:

- *Disaggregated Position Model:* The disaggregated position model (Tesch, 2016, 2018a) is a type of *event-based formulation*. Event-based formulations were first proposed for the multi-mode RCPSP (Zapata et al., 2018), before being adapted to the RCPSP (Koné et al., 2011). In event-based formulations, the time horizon is pre-decomposed into an ordered set of events. An event represents the start and end of one of the activities, without specifying which activity in particular. There are generally $\mathcal{O}(n)$ events — the RCPSP has the nice property that in semi-active schedules the start times of activities is at

0 or coincide with the end time of another activity and thus only n events would be required, however for different objectives this is not always the case and $2n$ events would need to be considered. The disaggregated position model considers two types of decision variables; (1) a binary variable for each activity and each combination of event pairs, which is equal to one if the activity starts at the first event and ends at the second event, and (2) a continuous variable for each event to represent the time that the event occurs. As the binary variable has three indices each in the order of n, the model has clearly $\mathcal{O}(n^3)$ variables. The nice property of this model though is that it does not contain the big-M parameters that are required by the $\mathcal{O}(n^2)$ models.

$\mathcal{O}(n^2)$:

- *Event-Based formulations:* Two event-based formulations for the RCPSP were initially proposed by Koné et al. (2011), one based on start and end variables and another based on on/off variables. In these models, two types of decision variables are used; (1) a binary variable that relates some status of the activities to events, and (2) a continuous variable for each event that indicate the time at which the event occurs. As the binary variables have two indices each in the order of n, the model has clearly $\mathcal{O}(n^2)$ binary variables. Event based formulations also suffer from relatively weak LP relaxations, due to the existence of big-M notation, however they are an interesting in that they too have quadratic number of variables and constraints with respect to the number of activities.
- *Sequence and Natural-Date Formulations:* These models use at least two types of variables: (1) binary variables are defined for every ordered pair of activities and are equal to one if the first activity precedes the second, and (2) continuous variables are defined for each activity to indicate the start time of that activity. Thus there are n continuous (natural-date) variables, $n(n-1)$ binary (sequencing) variables, and thus $\mathcal{O}(n^2)$ variables. These types of formulations typically have very weak LP relaxations as the constraints that link the binary and continuous variables require big-M notation where M is a function of the time-horizon. Formulations differ on how they enforce resource-feasibility:
  - *Minimal-Forbidden-Set-Based formulation:* A *forbidden set* is a set of activities that cannot be in progress simultaneously as they will exceed the capacity of one of the renewable resources. A *minimal forbidden set* is a

forbidden set such that any proper subset of the forbidden set is a feasible subset. Alvarez-Valdés and Tamarit (1993) proposed the first MIP model based on minimal forbidden sets. For each minimal forbidden set a constraint is added to ensure that at least one of the activities in the set precedes another activity in the set such that not all of the activities can be in progress simultaneously. Artigues et al. (2015) claim that due to the exponential number of constraints, they are not aware of any cases where this formulation has been used in practice.

  - *Flow-Based formulation:* The flow-based formulation, proposed by Artigues et al. (2003), is another example of a sequence and natural-date formulation. Instead of using the concept of minimal forbidden sets to account for resource feasibility, additional variables are introduced to account for the flow of resources between activities. More explicitly, for each ordered pair of activities and for each renewable resource, a continuous variable is introduced to indicate the amount of the resource that is transferred from the first activity to the second. For each activity and each resource, constraints are added to ensure the amount of flow into, and out of the activity must equal to the resource requirement of the activity. Likewise, constraints are added such that the amount of flow out of the dummy start activity and into the dummy end activity is equal to the resource capacity. Although this formulation still suffers from the weak LP relaxation, it is interesting as it was the first formulation of have a polynomial number of variables and constraints. It has also been shown to be competitive, compared with discrete-time formulations, for instances that have a large time horizon (Koné et al., 2011).

- *Hybrid Position flow model:* This model, introduced by Tesch (2016), can be seen as a hybrid of event-based and flow-based models. The number of events[1] is one more than the number of activities. At each event only one can start, and only one activity can complete. For each combination of events and each renewable resource a continuous variable is introduced to measure the amount of flow from the first event to the second. Tesch (2016) suggest the performance of this model is not competitive even for problem sizes of 30 activities.

[1]The term *events* are referred to as *positions* in the original document but we use events here for consistency

2.2.2 Limitations

To the best of our knowledge, in the more than 50 years of applying MIP to the RCPSP and its variants there does not exist a model for which the number of variables is better than quadratic with respect to the number of activities considered. In the literature the performance of the various MIP models are only compared on relatively small instances with between 30-120 activities. On instances with more activities, or large time horizons for the discrete-time formulations, Artigues et al. (2015) notes that even the time taken to load the models can become significant and suggests that techniques based on column-generation would need to be developed to improve this result. We are not aware of any such techniques at this time. In the next section we show how CP can be used to provide models with a linear number of variables and constraints and how they can naturally incorporate ideas from project scheduling.

2.3 Constraint Programming

Constraint Programming (Jaffar and Lassez, 1987; Marriott and Stuckey, 1998; Rossi et al., 2006) aims at solving *Constraint Satisfaction Problems* (CSPs) and *Constraint Optimisation Problems* (COPs). Like MIP, it is a problem-solving paradigm that establishes a clear distinction between two pivotal aspects of a problem: (1) a precise definition of the constraints that define acceptable solutions to the problem to be solved and (2) the algorithms and heuristics enabling the selections of decisions to solve the problem.

Scheduling has been one of the most successful application domains of CP (Laborie et al., 2018). There are two fundamentally different approaches for integrating scheduling into CP. The first approach complements classic CP on integer variables with a set of global constraints useful for modelling and solving scheduling problems. This approach has the advantage that they can be solved by general CP solvers, such as Chuffed, Gecode, CHIP, Opturion's CPX or Choco. The second approach is to develop a completely *scheduling-dedicated* modelling language on top of classical integer CP — this approach was taken by IBM's CP Optimizer, a complete redesign of ILOG Scheduler. In this section we provide an introduction to CP and demonstrate how the first approach is implemented. In the following section we provide an overview of the second approach.

2.3.1 Background

Constraint Satisfaction and Optimisation Problems

Informally a CSP is a problem that can be represented by a set of constraints that define acceptable assignments over a set of variables. Each variable has a *domain*, which defines the set of possible *values* to which the variable can be assigned. For the purpose of this thesis, domains can assumed to be a finite set of non-negative integers. A constraint determines which combinations of value assignments are acceptable and which are not. A solution to a CSP is an assignment of variables that satisfy all of the constraints.

More formally, a CSP is a tuple $P := (\mathcal{V}, \mathcal{C}, \mathcal{D})$ where $\mathcal{V}$ represents a set of variables, $\mathcal{D}$ a set of domains and $\mathcal{C}$ a set of constraints. Each variable $x \in \mathcal{V}$ is associated with a domain $dom(x) \in D$ of potential values. A *literal* l represents the assignment of a variable to a value, i.e., $x = v$, where $x \in \mathcal{V}$ and $v \in dom(x)$. For a literal l of the form $x = v$ the term $var(l)$ is used to denote the variable that has been assigned, i.e., in this case x. An *assignment* A over a set of variables $X \subseteq \mathcal{V}$ is a set of literals that has exactly one literal $x = v$ for each variable $x \in X$. An assignment A over $\mathcal{V}$ is called a *complete assignment*.

A constraint $C \in \mathcal{C}$ is defined over a set of variables, referred to as the scope of the constraint, and denoted $vars(C) \subseteq \mathcal{V}$. A constraint C specifies a set of *allowed* assignments over $vars(C)$. An assignment over $vars(C)$ that is not allowed is *disallowed* by C. The *projection* of an assignment A over a subset of variables X is defined as $\{l \in A : var(l) \in X\}$. An assignment A over $X \subseteq \mathcal{V}$ *satisfies* constraint C if $vars(C) \subseteq X$ and the projection of A over $vars(C)$ is allowed by C. An assignment A over $X \subseteq \mathcal{V}$ violates constraint C if $vars(C) \subseteq X$ and the projection of A over $vars(V)$ is disallowed by C. A *solution* of a CSP is a complete assignment which satisfies every constraint in $C \in \mathcal{C}$.

For consistency, throughout the thesis we will define integer variables, i.e. the typical decision variables considered in CP where the domains are sets of integers as

$$x \in \texttt{integer}(dom^{init}(x)),$$

where $dom^{init}(x)$ is the initial domain of the integer variable. Similarly, we define boolean variabies, i.e. integer variables whose domain is $\{0, 1\}$, explicitly as

$$b \in \texttt{boolean}.$$

An extension of a CSP is a COP. In an optimisation problem, the aim is to find an assignment A that satisfies all constraints and that optimises (minimises or maximises) some objective function. CP performs optimisation by first finding a solution that satisfies the constraints, i.e. the CSP is solved. It then adds an additional constraint to the model that ensures that any other solution that also satisfies the constraints will have a better objective value. This continues until the search tree is exhausted and no new solution is found, at which point the most recent solution obtained is proven to be optimal.

Constraint Propagation

Constraint propagation is the process of reducing a problem by eliminating from the domains of the variables those values that can never participate in a solution. Constraint propagation is fundamentally related to the notion of a *support*. A literal has a support with respect to a constraint if there exists an assignment of the remaining variables to values from their domains such that the constraint is satisfied. If a literal has no support with respect to some constraint then its value can be removed from the domain — we say that the value is *inconsistent*. Algorithms that determine which values are inconsistent with respect to constraints are known as *filtering algorithms* for the constraint. The filtering algorithms iterate until either none of them are able to prune a domain any more (we call that state a *fixpoint*), or one of the domains of variables become empty, i.e., propagation fails. This iterative process is called *constraint propagation*.

More formally, a CSP $P = (\mathcal{V}', \mathcal{C}', \mathcal{D})$ is a *reduced* form of a CSP $P = (\mathcal{V}, \mathcal{C}, \mathcal{D})$ if the same variables are considered, $\mathcal{V} = \mathcal{V}'$, and the domains of the variables are a subset of their corresponding domains, $dom(x)' \subseteq dom(x)$ for all $x \in \mathcal{V}$, and the set of solutions of P is the set of solutions to P'. The values removed from any $dom(x)$ are said to have been *pruned*.

In general, propagation is incomplete. That is, propagation may leave a value v in the domain of a variable x even though the literal $x = v$ does not participate in a solution. There are two reasons for why this is typical. Firstly, in CP constraints are considered separately and the domain reduction that relies on their conjunction is not inferred. Secondly, filtering algorithms for certain types of constraints can be too computationally expensive to prune all inconsistent values. To better understand this second point let us now introduce the different definitions of consistency.

Consistency

Local consistency conditions relate to the consistency of subsets of variables and constraints. Within the field of CP, there are a large variety of different types of consistency measures, many of which are not relevant to this thesis. Here, we briefly mention a few important local consistency measures. We make the distinction between the consistency of a variable and the consistency of a constraint. In general, a variable consistency measure will ensure that the current domain of the variable will satisfy certain properties for a specific subset of the constraints. The following two consistency measures are common examples of variable consistency.

- *Node consistency* considers *unary constraints*, i.e., constraints that only have a single variable in their scope. A variable is node consistent if all values within its domain are consistent with all unary constraints on the variable. Node consistency is typically performed in preprocessing.
- *Arc consistency* considers *binary constraints*, i.e., constraints that only have two variables in its scope. A variable $x \in \mathcal{V}$ is arc-consistent with respect to another variable $y \in \mathcal{V} : x \neq y$ if for every value in the domain of x there exists a value in the domain of y such that any binary constraint considering x and y is satisfied.

For more complicated constraints, i.e., for constraints that have more than two variables in its scope, it is often useful to talk about constraint consistency. A constraint consistency measure will ensure that the domains of the variables in the scope of the constraint satisfies certain properties. The following two consistency measures are common examples of consistency measures on constraints.

- *Generalised arc consistency* is an extension of arc consistency to constraints with more than two variables in their scope. A variable $x \in \mathcal{V}$ is generalized arc consistent (GAC) with respect to a constraint if every value in the domain of the variable can be extended to all the other variables of the constraint in such a way that the constraint is satisfied. A constraint is GAC if all variables is its scope are GAC consistent with respect to the constraint.
- *Bounds consistency* is commonly defined with respect to an interval support. Value $v \in dom(x)$ of variable x has an interval support with respect to constraint C if there exists a feasible assignment to C where $x = v$ and $y \in [min(dom(y)), max(dom(y))]$, for every other variable $y \in vars(C) \setminus \{x\}$. A constraint is bounds consistent if for each variable $x \in vars(c)$ each of the

values $min(dom(x))$ and $max(dom(x))$ has an interval support in C. [2]

In the domain of scheduling even bound consistency is usually too expensive (Vilím, 2007). In practice, global constraints are defined which perform propagation according to certain rules. Next we introduce the concept of global constraints and then some important rules for scheduling.

2.3.2 Global Constraints

One of the strengths of CP is in the ability to define global constraints. The power of global constraints is threefold. Firstly, global constraints have the self-evident advantage of additional expressiveness, i.e., they can more clearly communicate the underlying purpose of the constraint. Secondly, global constraints allow large sets of constraints to be represented in a compact form, which can reduce the number of constraints required by orders of magnitude. Thirdly, and perhaps most importantly, they enforce local consistency for the whole set of constraints, which is superior to enforcing consistency for each single constraint individually. Filtering algorithms for a global constraint must make reasonable (polynomial) time and space complexity and hence there is of course a limitation to how many different types of constraints can be grouped within the same global constraint.

In this section we introduce three global constraints; `AllDifferent`, `Disjunctive`, and `Cumulative`. The `AllDifferent` constraint is the quintessential global constraint, for which an GAC algorithm was first proposed by Regin (1994) based on matching theory. The remaining two global constraints arise in scheduling problems, where `Disjunctive` is a special case of `Cumulative`. Even determining whether `Disjunctive`, and thus `Cumulative`, is satisfiable is NP-Complete and therefore it is NP-hard to enforce bounds consistency on these constraints (Fahimi et al., 2018). To overcome this there have been many rules proposed for which a weaker notion of consistency than bounds consistency can be achieved for these global constraints in polynomial time. A full review of these rules and corresponding algorithms is outside the scope of this thesis. Instead we satisfy ourselves by introducing two well-known rules for the `Cumulative` constraint: *TimeTable* and *Edge-Finding*. We discuss these rules with respect to the `Cumulative` constraint particularly as this constraint is used to model our reference problem, the RCPSP/max. There exist other filtering algorithms for the `Cumulative` constraint such as Ener-

[2] Although not a focus of the thesis, we note that some papers in the literature (Loong et al., 2016) differentiate bounds consistency on whether the interval supports are considered to be strictly integer, known as $\mathbb{Z}$-*bounds consistency*, or allowed to be continuous, known as $\mathbb{Q}$-*bounds consistency*. Unless otherwise stated we assume that interval supports are integer.

getic Reasoning (Erschler and Lopez, 1990), Not-First/Not-Last (Nuijten and Aarts, 1996) and hybrids such as Precedence Energetic Reasoning (Laborie, 2003a) and *Energetic Edge-Finding* (Tesch, 2018b), yet due to their higher computation complexities or need for more iterations to achieve equivalent levels of filtering they tend to be less useful in practice.

AllDifferent

The classical example of a global constraint is the `AllDifferent` constraint that is posted on a set of variables and amalgamates a full network of inequality constraints, i.e.,

$$\texttt{AllDifferent}([x_i]_{i \in X}) \iff x_i \neq x_j \; \forall x_i, x_j \in X : i < j$$

A great example given by Lombardi (2009) to demonstrate how enforcing local consistency on the global constraint is superior to enforcing local consistency for the whole set of constraints is as follows.

Example 2.3.1. Consider the following CSP

$$x_0 \neq x_1, x_0 \neq x_2, x_1 \neq x_2$$
$$x_0, x_1 \in \texttt{integer}([1,2]), x_2 \in \texttt{integer}([1,2,3])$$

It can be seen that enforcing arc consistency on each constraint individually that no domain can be reduced, as for each x_i all values v in the domain D_i have a support in D_j, in the context of each of the three constraints. In contrast, by considering all constraints at the same time, it is possible to see that values 1 and 2 must be assigned to constraints x_0 and x_1 in order to have a feasible solution, and thus the domain of x_2 can be reduced to $D_2 = \{3\}$.

Disjunctive

The `Disjunctive` and `Cumulative` global constraints are typically used in non-preemptive resource-constrained scheduling problems, where the variables X represent the start time of the activities $\mathcal{V}$. Variants of these constraints exist that account for optional activities, a special type of execution mode where the activity does not have to be processed, or when pre-emption is allowed. Here we simply introduce the non-preemptive compulsory versions of the constraints. `Disjunctive` can be seen as a decision variant of the scheduling problem $1|r_i|L_{max}$, which is

known to be strongly NP-hard (Lenstra et al., 1977). The objective is to minimize the maximum lateness $L_{max} = \max_{i=1}^{n} \max\{0, x_i + d_i - due_i\}$ where $L_{max} = 0$ iff the instance is feasible. The constraint is defined as follows,

$$\texttt{Disjunctive}([x_i]_{i\in X}, [d_i]_{i\in X}) \iff x_i + d_i \leq x_j \vee x_j + d_j \leq x_i \quad \forall i, j \in X$$

where X is an array of activity start variables, and $[d_i]_{i\in X}$ the corresponding durations. The release and due dates are built into the domains of each activity variable.

There exist a number of rules such as Edge Finding, Not-First/Not-Last, and Detectable Precedences for `Disjunctive` that can be computed in $\mathcal{O}(n\ log(n))$ (Vilím, 2007), whereby a number of these constraints would require $\mathcal{O}(n^2)$ complexity for the more general `Cumulative` constraint. For that reason it is worthwhile introducing `Disjunctive` separately.

Cumulative

The `Cumulative` global constraint, which was first introduced by Aggoun and Beldiceanu (1993), models the renewable resources for the working example the RCPSP/max. The `Cumulative` global constraint aims to satisfy the cumulative scheduling problem, which can be seen as a decision variant of the scheduling problem $P|r_j|L_{max}$, i.e., similar to `Disjunctive` but now where a single renewable resource $k \in \mathcal{R}$ is considered as opposed to a single disjunctive resource. The constraint is defined as follows,

$$\texttt{Cumulative}([x_i]_{i\in X}, [d_i]_{i\in X}, [r_{i,k}]_{i\in X}, R_k) \iff \sum_{i\in X} \sum_{x_i \leq \tau < x_i + d_i} r_{i,k} \leq R_k \ \ \forall t \in H$$

where $[x_i]$ is an array of activity start variables, and $[d_i]$ and $[r_{i,k}]$ provide the corresponding durations and resource requirements respectively. The following rules are defined for which polynomial filtering algorithms exist.

Time-table The core idea of *timetabling* is to track activities for which their latest start time is strictly less than their earliest end time, i.e., $i \in V$ such that $LST_i < EET_i$. Although the start time of these activities are not necessarily fixed, these activities require the resource during the interval $[LST_i, EET_i]$ - these intervals are

known as *compulsory parts*. By aggregating the compulsory parts a *resource profile*, also known as a timetable, is computed. This resource profile is maintained during the search and used to both detect infeasibility and to update time bounds for activities both with a complexity of $\mathcal{O}(n\ log(n))$ (Lahrichi, 1982). For each task the earliest start is updated to the first possible time point with respect to the resource profile such that the capacity is not exceeded.

More formally let, $RQ(t)$ be the resource profile at time $t \in H$ and $RQ(t,i)$ be the resource profile in the hypothesis that activity i is running at time $t \in H$, i.e., $RQ(t,i) = RQ(\tau)$ if $LST_i \leq \tau < EET_i$ and $RQ(\tau) + RQ(\tau) + r_{i,k}$ otherwise. If for any activity $i \in V$ the following rule is true,

$$RQ(\tau, i) > C_k, \forall \tau \in [EST_i, ..., LST_i] \tag{2.2}$$

then it is possible to update the earliest start time of i to the following

$$EST_i = \min\{\tau | RQ(\tau', i) \leq C_k, \forall \tau' \in [\tau, ..., \tau + d_i]\} \tag{2.3}$$

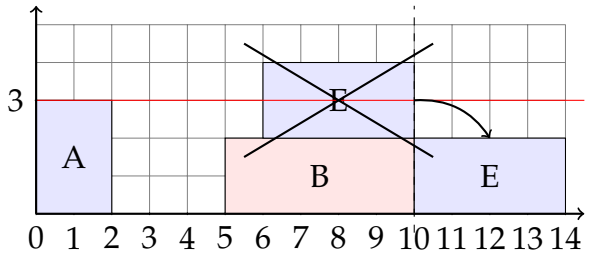
In the following example we show how time-tabling can infer information about the resource profile before the starting times of activities are assigned to a specific value.

Example 2.3.2. We consider the instance of the RCPSP/max considered in Example 2.1.1, but now with a resource capacity of 3 (instead of 4). We consider the point during search / propagation where the earliest and latest start time are as in the table in Figure 2.4. Hence activity A has been assigned to start time 0, the earliest start time of B and C have been updated to 2, and 6 respectively due to the minimum time lags. The maximum time lag between A and C ensure the latest start time of C is 9, and again then due to the minimum between B and C, the latest start time of B is 5. The earliest start time of activity D is updated to 2 — this is in fact done by the time-table constraint however as A is fixed this propagation is straight-forward. The minimum time lag between D and E propagates to ensure the earliest start time of E is 6.

As the difference between the latest start and earliest start of activity B is less than the duration of the activity, information can be assumed about its contribution to the resource profile. More explicitly, activity B has a compulsory region between time 5 and 10, as shown by the red box in Figure 2.4. As EST_E is currently 6, due to the inferred resource usage of activity B, the time-table constraint propagates $EST_E = 10$ as shown in Figure 2.4. Propagation continues by updating $EST_D = 6$

due to the maximum time lag between D and E. Again the time-table constraint is used to update $EST_D = 10$, and then due to the minimum time lag between D and E, $EST_E = 14$. This is now a fix-point.

Figure 2.4: An example (from Example 2.3.2) of how time-table filtering can update EST_E from 6 to 10.



i	EST_i	LST_i
A	0	0
B	2	5
C	6	9
D	2	T
E	6	T

The main limitation of time-tabling is that nothing is propagated until the domains of activity start time variables are so small that the compulsory regions can be inferred. This means that unless strong commitments are made early in the search, this approach is not able to propagate efficiently.

(Extended) Edge-Finding Edge-finding reasons over the *energy* of sets of activities, $\Omega \subseteq V$. The energy of activity $i \in V$ for resource $k \in R$ is defined as the duration of the activity multiplied by the resource required by the activity, i.e., $e_i = d_i \cdot r_{i,j}$. The earliest start (latest completion) time of the set Ω is considered to be the earliest start (latest completion) time of any activity in the set, i.e., $EST_\Omega = \min_{i\in\Omega}\{EST_i\}$ and $LET_\Omega = \max_{i\in\Omega}\{LET_i\}$. The energy of the set is considered to be the energy of each of the activities in the set, i.e., $e_\Omega = \sum_{i\in\Omega} e_i$. The total energy available for Ω is $R_k(LET_\Omega - EST_\Omega)$. If any subset Ω requires more energy than there is available, there is no solution.

If for an activity $i \in V$ and a subset of activities $\Omega \subseteq V \setminus \{i\}$ the following check is true,

$$e_\Omega + e_i > R_k(LET_\Omega - EST_\Omega) + \sigma^j_\Omega \tag{2.4}$$

then is possible to update the earliest start time of i to the following

$$EST_i := \max_{\Omega'\subset\Omega : rest(\Omega', r_{ik}) > 0)} (EST_{\Omega'} + \lceil \frac{rest(\Omega', r_{ik})}{r_{ik}} \rceil) \tag{2.5}$$

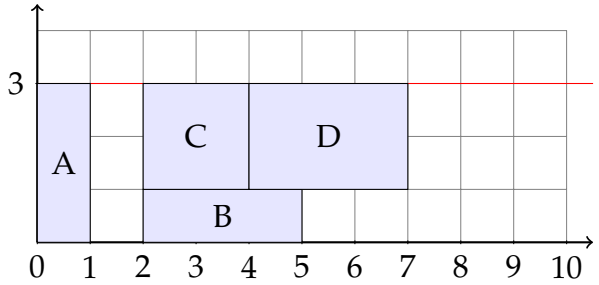
Where $\sigma^i_\Omega = r_{ik} \cdot \max(EST_\Omega - EST_i, 0)$ is the maximum amount of resource

used by activity i before EST_Ω and $rest(\Omega', r_{ik}) = e_{\Omega'} - (R_k - r_{ik}) \cdot (LET_{\Omega'} - EST_{\Omega'})$.

The first complete algorithm for extended edge-finding was introduced by Mercier and Van Hentenryck (2007) with a complexity of $\mathcal{O}(k\, n^2)$, where k is the number of distinct resource requirements of the activities. This complexity was improved to $\mathcal{O}(k\, n\, log(n))$ by Vilím (2009).

Example 2.3.3. Consider the problem with a single resource k with capacity $R_k = 3$ and four activities $V = \{A, B, C, D\}$ with earliest start time, latest end time, duration and resource requirement as given in Figure 2.5. Consider activity D with respect to set $\Omega = \{A, B, C\}$. Hence $EST_\Omega = 0$, $LET_\Omega = 5$ and $e_\Omega = 3 + 3 + 4 = 10$. It is possible to see that check (2.4) would be true for Ω and D as $e_\Omega + e_D = 10 + 6 = 16 > R_k(LET_\Omega - EST_\Omega) + \sigma^j_\Omega = 3(5 - 0) + 0 = 15$. Thus the start time of activity D can be updated according to Equation 2.5 to $EST_D = 4$ where the subset corresponding to the maximum increase is $\Omega' = \{B, C\}$. It is possible to see that the time-tabling would not propagate on this example.

Figure 2.5: An example of how edge-finding filtering can update EST_D from 0 to 4



i	EST_i	LET_i	d_i	r_{ik}	e_i
A	0	5	1	3	3
B	2	5	3	1	3
C	2	5	2	2	4
D	0	∞	3	2	6

Time-Table Edge-Finding As observed by Vilím (2011) edge finding does not dominate timetabling and vice versa, therefore edge finding can be improved by taking into account the timetable. In order to use the timetable each activity is split into a *fixed part* accounted for in the timetable and a remaining *free part*. This filtering algorithm can be seen as a hybrid of time tabling and edge finding and is hence referred to as *time-table edge finding*. The additional strength of propagation comes at a higher worst-case complexity of $\mathcal{O}(n^2)$. Time-table edge finding has predominantly had success at improving lower bounds; when it was first introduced by Vilím (2011) it improved 169 out of the 438 open RCPSP problems (in 9 cases backtrack free) and then when combined with no-good learning by Schutt et al. (2013b) a further 60% of the instances had their lower bounds improved.

2.3.3 Modelling the RCPSP/max

For continuity we will now briefly mention how our reference problem, the RCPSP/max, is typically modelled in CP. The starting time of each activity $i \in \mathcal{V}$ is assigned a decision variable x_i, which must be mapped to an integer from the initial domain $D^{init} := [EST_i, LST_i]$. Initial domains are typically obtained through determining a maximum time horizon T based on a heuristic solution or conservative assumption and then considering the precedence graph. A typical CP model for the RCPSP/max is then,

$$\begin{aligned}
\min \quad & \max_{i \in \mathcal{V}} (x_i + d_i) && && \text{(2.6a)}\\
\text{s.t.} \quad & x_i + \delta_{i,j} \leq x_j && \forall (i,j) \in A && \text{(2.6b)}\\
& \texttt{Cumulative}([x_i], [d_i], [r_{i,k}], R_k) && \forall k \in R && \text{(2.6c)}\\
& x_i \in \texttt{integer}([EST_i, ..., LST_i]) && \forall i \in V && \text{(2.6d)}
\end{aligned}$$

The objective function (2.6a) is to minimise the maximum completion time of any of the activities. Constraints (2.6b) enforce the generalised precedence constraints. Each resource is represented by a single `Cumulative` constraint (2.6c) with a constant capacity over the considered project duration. The propagation algorithm(s) used to implement the global constraints are typically not specified at the time of modelling. Each variable must be assigned to a value from its original domain (2.6d). Note that the number of decision variables is equal to the number of activities, the number of precedence constraints is equal to the number of generalised precedence constraints, and the number of cumulative constraints is equal to the number of renewable resources. Assuming that the number of precedence constraints is proportional to the number of activities, this model is linear with respect to both the number of variables and constraints.

We will now introduce some typical methods that CP uses to search and prove optimality for scheduling problems.

2.3.4 Search

Constraint propagation rarely reduces the domains of variables of a problem to a single value. In most cases, it is necessary to reduce the problem further by way of search, i.e., by exploring the different values in the domain of the variables. A backtracking tree search is used that branches by assigning values to variables,

i.e., on the left branch a literal is chosen $x_i = d_i$ and on the right branch the value from the literal is removed from the domain of the variable, $D'_i = D_i \setminus \{d_i\}$. Once a branching decision has been made, constraint propagation is run to propagate the implications of that decision, thus further reducing the domains of the unfixed variables. If some constraint is violated by the partial assignment, then there do not exist solutions within the subtree and the solver backtracks. This kind of inference can dramatically reduce the amount of search that CP solvers must do compared to naive enumeration approaches, and is one of the core strengths of CP. The search is complete because every solution of the problem will be found, and if no solution is found then the problem is known to not have a solution.

Variable-Value Ordering

Although generic search algorithms in CP such as impact-based search (Refalo, 2004), weight-degree heuristics (Boussemart et al., 2004) and activity-based search (Michel and Van Hentenryck, 2012) have had some success, in general the user must tell CP solvers how to search, i.e., rules for deciding which variables and values to branch on. Furthermore the order in which variables and their values are selected can have a dramatic effect on how much of the search space needs to be explored.

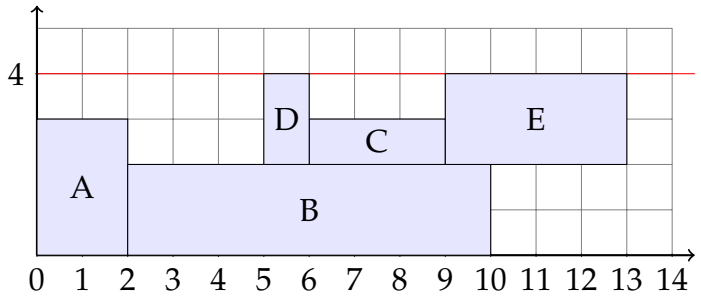
For scheduling problems with regular objective functions such as the RCPSP and RCPSP/max, assigning variables to their lower bound, i.e., fixing the start time of activities to their earliest starting time generates active schedules and thus is the default choice for value selection. For irregular objective functions the choice of value selection is not trivial.

There are many parallels between the choice of variable selection and priority rules and activity lists introduced with respect to schedule generation schemes. In the following example we demonstrate how an activity list is implemented in CP and highlight how constraint propagation (using time-tabling to filter the `Cumulative` constraint) can avoid areas of search that would be explored by the scheduling generation scheme with unscheduling step.

Example 2.3.4. In this example we implement the same list scheduling algorithm as in Example 2.1.3 but now just to guide our variable selections within CP search. Figure 2.6 shows the partial schedule and current bounds after the first three choice points, where activity A is assigned to 0, B to 2 and C to 6, equivalent to Example 2.1.3. Now, due to the time-table propagator, the earliest start time of activity E is

updated to 9, and then due the maximum time lag between D and E, $EST_D = 5$. This contrasts significantly to how the list scheduling algorithm used in Example 2.6 assigns the start time of D to 2, only to unschedule the activity later in the search. The algorithm then assigns the start time of activity D to 5, and then E to 9.

Figure 2.6: The schedule obtained by using the activity list $\ell = \{A, B, C, D, E\}$ to determine the variable ordering to guide a CP search. The schedule is obtained without obtaining a single fail, this contrasts to how the schedule generation scheme from Example 2.3 requires an unscheduling step.



Schedule-or-Postpone

Schedule-or-Postpone is a tree-based strategy that relies on constraint propagation to determine *dominance relations* during search for scheduling problems. A dominance relation is a relationship between two possible solutions to a problem where one solution is always preferred to another. The type of dominance relations considered by Schedule-or-Postpone are those where if one solution is infeasible then so too must the other one. The method was first proposed for the RCPSP with regular objective functions as a method of only creating active schedules (Pape et al., 1994), and thus much smaller search trees than naive variable-value ordering.

The basic Schedule-or-Postpone strategy is as follows. At each fix-point, an activity for which all of its predecessors have been scheduled is selected according to some rule. This creates a choice point in the search tree. On the left branch the chosen activity is scheduled to its earliest start time. On the right branch the activity is marked is non-selectable, i.e., the activity is postponed. If the earliest start time of the activity is modified by propagation, the activity is marked as selectable again.

The Schedule-or-Postpone strategy is the foundation of the *SetTimes* algorithm, first proposed by (Pape, 1994) and recounted in detail by Godard et al. (2005). In SetTimes, activities are selected in a chronological way, i.e., the activity with the lowest earliest start time is selected first, with lowest latest end time and then activity index used to break ties. CP Optimizer uses many variants and extensions

of the SetTimes algorithm, for example, Laborie and Rogerie (2016) recently introduced a generalisation in which the selected activity is set to an indicator value, which is not necessarily the earliest start time, an approach beneficial for problems with irregular objective functions. Yet the idea of using the Schedule-or-Postpone strategy to determine dominance relations is the same.

In some particular cases, such as for the RCPSP, the Schedule-or-Postpone strategy, and thus SetTimes, is known to be complete in the sense that it will find an optimal solution if there is one (Pape et al., 1994). However in many cases, such as even for the RCPSP/max, this is not the case, as activities can no longer be continuously right-shifted until they can be scheduled. To overcome this limitation, weaker versions of the Schedule-or-Postpone strategy are known whereby if all activities in the decision set are marked unselectable, activities can become selectable again by removing their earliest start time from their current domains. These weaker versions are clearly not as beneficial at closing off search trees, and thus proving optimality, however they can still be exceptionally useful at generating good solutions, as is demonstrated in the following example for the RCPSP/max.

Example 2.3.5. The aim of the example is to highlight the dominance relations that can be inferred through the Schedule-or-Postpone strategy. In this example we consider how the SetTimes algorithm can find a feasible schedule on the instance of the RCPSP/max given in Example 2.1.1. Initially the two eligible activities are A and D, both have the same earliest start time, however the latest start time of activity A is one less than D, and so A is selected and scheduled to its earliest start time, and the constraints are propagated. At the next choice point, activities B and D are eligible, both have an earliest start time of 2, however the latest start time of B is 5 due to the maximum time lag between A and C and the minimum time lag between B and C, and so B is selected and assigned to 2. At the next choice point, D and C are both eligible, however the earliest start time of D is 2 compared to 6 for E, and so D is chosen and assigned to 2. However the propagation of this value assignment leads to a failure: activity E is fixed to start at 6 and then the time-table constraint propagates to try and set the earliest start time of C to be at 10 however this is after the latest start time.

At a failure the algorithm backtracks and sets D as "unselectable". Now only activity C is eligible, and thus C is fixed to its earliest start time of 6. The time-table constraint propagates the earliest start of E to be at 9, and then the maximum time lag between D and E propagates the earliest start time of D to 5. As the value of 2 is filtered from the domain of D, the status of D is now updated to "selectable".

Hence the algorithm continues by fixing D to 5, which propagates to fix E to 9.

Now consider what would have occurred if the Schedule-or-Postpone strategy was not used. After the failure that occurs when D is fixed to 2, the algorithm would move to the right branch of the tree where $EST_D > 2$. Again D would be selected and now assigned to 3, its current earliest start time. Again a failure would occur, and then the algorithm would move to the right branch and again D would be selected as it has the earliest start time 4, which again would fail. The Schedule-or-Postpone strategy effectively allowed us to jump over all of these failures.

In the next section we will introduce an alternative method of determining dominance relations during search.

Lazy Clause Generation

Recently there have been a number of developments in CP solvers that incorporate methods developed for the Boolean Satisfiability Problem (SAT). SAT was the first problem proven to be NP-Complete (Cook, 1971) and thus lies at the heart of the theory of computational complexity. SAT considers a set of boolean variables, $\mathcal{B}$. With respect to SAT, a *literal*, l, is a boolean variable $b \in \mathcal{B}$, i.e., $l \equiv b$, or its negation, i.e., $l \equiv \neg b$. A *clause*, c, is a set of literals in disjunction, i.e., $\{l_1 \vee ... \vee l_n\}$. Thus a clause is satisfied if at least one of its literal l_i is true. An *assignment*, $\mathcal{A}$, is a set of literals that does not include a variable and its negation, $\nexists b \in \mathcal{B} : \{b, \neg b\} \subseteq \mathcal{A}$. A theory $\mathcal{T}$ is a set of clauses. A SAT problem consists of finding an assignment $\mathcal{A}$ containing each boolean variable in either a positive and negative context, and satisfying all clauses in the theory $\mathcal{T}$.

SAT solvers propagate clauses by *unit-propagation*. Unit propagation detects failure if all literals in a clause are false and detects a new unit consequence l if all but one literals in a clause are false, i.e., $c \equiv l \cup c'$ and $\{\neg l' | l' \in c'\} \in \mathcal{A}$, in which case it adds l to the current assignment $\mathcal{A}$. Unit propagation continues until failure is detected, or no unit consequences can be determined, i.e., a *fixed point* is reached. If a fixed point is reached a solver chooses an unfixed variable $b \in B$ and creates two separate problems $(\mathcal{T}, \mathcal{A}' \cup \{b\}), (\mathcal{T}, \mathcal{A}' \cup \{\neg b\})$. This literal added to the assignment by choice is referred to as a *decision literal*.

Modern SAT solvers contain effective methods that aim to prevent search from revisiting similar parts of the search space. More explicitly, the SAT solvers records the clause that causes unit propagation, also known as the *explanation*, for each unit consequence discovered to determine a set of mutually incompatible decisions, known as a *nogood*. Nogoods can be added as a new clause to the theory of the

problem and can drastically reduce the size of the search space that needs to be examined. The autonomous search procedures in modern SAT solvers are also very powerful. These search procedures concentrate on the variables that are involved in the most failures, which the solver keeps track of. A significant limitation of SAT is the often huge models that are required to represent a problem using only boolean variables and clauses.

Lazy Clause Generation (LCG) (Ohrimenko et al., 2009) is a method used to help CP solvers take advantage of the nogood learning and activity based search of modern SAT solvers, while attempting to overcome the issue of the huge models required to encode the entire problem as clauses over boolean variables. The key idea of LCG is to lazily create a clausal representation of the problem that a SAT solver can understand by explaining the propagation achieved by constraints in CP. Thus in LCG a constraint propagator is no longer just a mapping from domains to domains, it is also a generator of clauses describing propagation.

In LCG, each integer variable in the CP problem has a clausal representation in the SAT solver. We denote boolean variables with double square brackets, $[\![.]\!]$. Each integer variable $x \in X$ with initial domain $D_{init}(x) = [l, ..., u]$ is represented by $2(u-l)+1$ boolean variables: $[\![x = l]\!], [\![x = l+1]\!], ..., [\![x = u]\!]$ and $[\![x \leq l]\!], [\![x \leq l+1]\!], ..., [\![x \leq u-1]\!]$. The boolean variable $[\![x = d]\!]$ is *true* if variable x takes the value d, and *false* if x takes a value different from d. Similarly, the variable $[\![x \leq d]\!]$ is *true* if x takes a value less or equal to d and *false* otherwise. For simplicity we use the notation $[\![x \geq d]\!]$ to refer to the literal $\neg[\![x \leq d-1]\!]$.

To ensure the assignment of boolean variables are consistent with the integer variable x, a number of clauses are added to the SAT solver. These constraints are known as the *domain clauses* and described as follows:

$$[\![x \leq d]\!] \rightarrow [\![x \leq d+1]\!] \qquad \forall d \in [l, ..., u-1) \tag{2.7a}$$

$$[\![x = l]\!] \leftrightarrow [\![x \leq l]\!] \tag{2.7b}$$

$$[\![x = d]\!] \leftrightarrow ([\![x \leq d]\!] \wedge \neg[\![x \leq d-1]\!]) \qquad \forall d \in (l, ..., u) \tag{2.7c}$$

$$[\![x = u]\!] \leftrightarrow \neg[\![x \leq u-1]\!] \tag{2.7d}$$

Typically constraints (2.7*b* − 2.7*d*) are added lazily when propagation needs to express something using the literal $[\![x = d]\!]$ (Feydy, 2010). Any assignment on the boolean variables can be converted back to the domains of the CP integer variables.

It is important to point out that LCG impacts the number of variables used to encode the RCPSP(/max). As a pure CP approach uses a single integer variable to represent the start time of each of the activities the number of variables of the model is $\mathcal{O}(n)$. Now that a boolean variable is used to encode each element of the initial domain of each of the integer variables this is essentially a discrete-time representation of the problem, and thus the number of variables is $\mathcal{O}(n \cdot T)$. Therefore CP solvers the utilise LCG will inevitably have a issue with scalability as the time-horizon increases.

When a propagator f changes the domain of the variables, i.e., $f(D) \neq D$, we assume that the propagator can determine a clause c to explain each domain change. An *explanation*, $c \rightarrow l$, is a conjunction of literals, c, true in the current assignment that result in a literal, l, that was previously unfixed in the current assignment to become true. The explanation clauses of the propagation are sent to the SAT solver on which unit propagation is performed.

The nogood generation used in LCG is based on an *implication graph* and the *first unique implication point* (1UIP). The implication graph is a directed acyclic graph where nodes represent fixed literals and arcs represent the reason why a literal became true, i.e., the explanations. Every time a literal is fixed through unit-propagation, a node is added to the implication graph along with the arcs from the literals (nodes) that explain the assignment. When a literal is fixed by a search decision only the node is added to the graph, i.e., the node will have no incoming arcs, and a counter known as the *search level* is incremented. Every node and arc is associated with the search level at which they are added to the graph.

Once propagation leads to a failure, the 1UIP nogood is calculated based on the implication graph. A conflict occurs when the unit propagation reaches a clause where all literals are false. This clause is the starting point of the analysis and builds a first tentative nogood. Literals in the tentative nogood are replaced one by one by the literals from their incoming edges in the implication graph, in reverse order of their addition to the graph. This process continues until the tentative nogood contains exactly one literal from the current search level. The resulting nogood is called the 1UIP nogood (Moskewicz et al., 2001). The solver then backtracks, undoing the previous decision and its consequences, and the 1UIP nogood is added as a clause to the theory, which will force unit propagation. In the following example we demonstrate how LCG can determine the same dominance relation as determined by the Schedule-or-Postpone strategy in Example 2.3.5.

Example 2.3.6. This example is modified version of one contained in Schutt et al.

(2013c) to demonstrate how non-trivial clauses can be learnt through LCG. Consider the instance of the RCPSP/max introduced in Example 2.1.1 and assume an initial time horizon of $T = 20$. Hence after the initial propagation of the precedence constraints, the domains are $D(S_A) = [0, ..., 10], D(S_B) = [2, ..., 12], D(S_C) = [6, ..., 17], D(S_D) = [0, ..., 12]$, and $D(S_E) = [4, ..., 16]$.

Search now sets $S_A \leq 0$. This sets literal $[\![S_A \leq 0]\!]$ as true and unit propagation on the domain clauses sets $[\![S_A \leq 1]\!]$, $[\![S_A \leq 2]\!]$, etc. In the remainder of this example we ignore the propagation of the domain clauses unless they are particularly relevant to other propagation.

The maximum time lag $S_C - 9 \leq S_A$ forces $S_C \leq 9$ with explanation $[\![S_A \leq 0]\!] \rightarrow [\![S_C \leq 9]\!]$. The minimum time lag $S_B + 4 \leq S_C$ forces $S_B \leq 5$ with explanation $[\![S_C \leq 9]\!] \rightarrow [\![S_B \leq 5]\!]$. The timetable propagator then forces $S_D \geq 2$, with explanation $[\![S_A \leq 0]\!] \rightarrow [\![S_D \geq 2]\!]$. The minimum time lag $S_D + 4 \leq S_E$ forces $S_E \geq 6$ with explanation $[\![S_D \geq 2]\!] \rightarrow [\![S_E \geq 6]\!]$. Next the search sets $S_B \leq 2$. No propagation other than the propagation of domain clauses for B occur at this point.

Next the search sets $S_D \leq 2$. The maximum time lag $S_E - 4 \leq S_D$ forces $S_E \leq 6$ with explanation $[\![S_D \leq 2]\!] \rightarrow [\![S_E \leq 6]\!]$. Lets now consider the compulsory parts of the resource profile resulting from B and E. As both activities are now effectively fixed, 2 units of resource is required by B from $[2, ..., 10)$ and 2 units are required by E from $[6, ..., 10)$, hence there is 0 resource available between $[6, ..., 10)$. As the initial propagation of the precedence constraints ensure $S_C \geq 6$, the timetable constraint can filter the domain of S_C. Here we consider pointwise explanation for domain filtering of the time-table propagator as defined in Schutt et al. (2011) [3]. Considering the time point 8, the timetable propagator forces $S_C \geq 9$ with explanation $[\![S_E \leq 8]\!] \wedge [\![S_E \geq 5]\!] \wedge [\![S_B \leq 8]\!] \wedge [\![S_B \geq 1]\!] \wedge [\![S_C \geq 6]\!] \rightarrow [\![S_C \geq 9]\!]$.

The previous propagation forces a compulsory part of C to occur at 9 which causes the demand of the resource to exceed the capacity at this time. A pointwise explanation of this failure [4] is $[\![S_B \leq 9]\!] \wedge [\![S_B \geq 2]\!] \wedge [\![S_C \leq 9]\!] \wedge [\![S_C \geq 9 \wedge [\![S_E \leq 9]\!] \wedge [\![S_E \geq 6]\!]]\!] \rightarrow failure$. The relevant parts of the implication graph are shown in Figure 2.7.

The nogood generation process starts from the original explanation of failure. It creates a tentative nogood by replacing the failure node with its explanation.

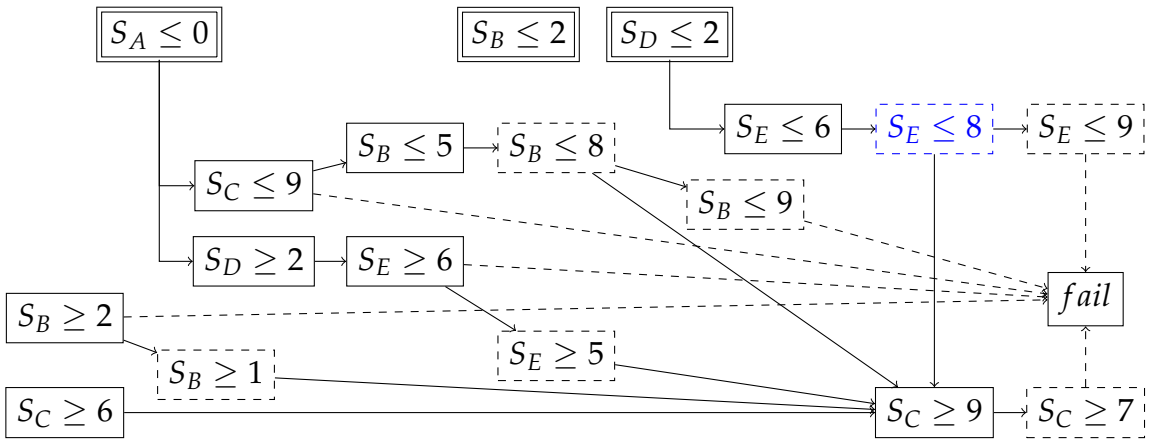
[3] The pointwise explanation for filtering the time-table constraint is $([\![t + 1 - d_j \leq S_j)]\!] \wedge \bigwedge_{i \in \Omega} [\![t + 1 - d_i \leq S_i]\!] \wedge [\![S_i \leq t]\!]) \rightarrow [\![t + 1 \leq S_j]\!]$, Ω are the activities that compulsory are used at time point t, and j is the activity whose domain is being filtered.

[4] The pointwise explanation for consistency checking of the time-table constraint is $\bigwedge_{i \in \Omega} [\![t + 1 - d_i \leq S_i]\!] \wedge [\![S_i \leq t]\!] \rightarrow false$, where Ω are the activities that compulsory are used at time point t.

Hence the tentative nogood is initially the clauses that explain the failure, i.e., $[\![S_E \leq 9]\!] \wedge [\![S_E \geq 6]\!] \wedge [\![S_B \leq 9]\!] \wedge [\![S_B \geq 2]\!] \wedge [\![S_C \leq 9]\!] \wedge [\![S_C \geq 9]\!]$. However as $[\![S_C \geq 7]\!]$ and $[\![S_E \leq 9]\!]$ are both associated with the current conflict level, they are replaced by their explanations. Hence $[\![S_C \geq 7]\!]$ is replaced in the nogood by $[\![S_C \geq 9]\!]$ and $[\![S_E \leq 9]\!]$ is replaced by $[\![S_E \leq 8]\!]$. Again both of the new clauses are associated with the current conflict level. Hence $[\![S_C \geq 9]\!]$ is replaced by its explanation, i.e., $[\![S_B \leq 8]\!] \wedge [\![S_B \geq 1]\!] \wedge [\![S_C \geq 6]\!] \wedge [\![S_E \leq 8]\!] \wedge [\![S_E \geq 5]\!]$. Now $[\![S_E \leq 8]\!]$ is the only clause in the nogood from the current conflict level and hence the current tentative nogood is the 1UIP nogood. Removing the implied clauses from this nogood, the 1UIP nogood is $[\![S_B \leq 9]\!] \wedge [\![S_B \geq 1]\!] \wedge [\![S_C \leq 9]\!] \wedge [\![S_C \geq 6]\!] \wedge [\![S_E \leq 8]\!] \wedge [\![S_E \geq 5]\!]$. Rewritten as a clause it is $(\neg[\![S_B \leq 9]\!] \vee [\![S_B \leq 0]\!] \vee \neg[\![S_C \leq 9]\!] \vee [\![S_C \leq 5]\!] \neg[\![S_E \leq 8]\!] \vee [\![S_E \leq 4]\!])$.

Now the solver backtracks to the previous decision level, undoing the decision $S_D \leq 2$ and its consequences. The newly added nogood unit propagates to force $S_E \geq 9$ with explanation $[\![S_B \leq 9]\!] \wedge [\![S_B \geq 1]\!] \wedge [\![S_C \leq 9]\!] \wedge [\![S_C \geq 6]\!] \wedge [\![S_E \geq 5]\!] \rightarrow [\![S_E \geq 9]\!]$. The precedence constraint $S_E - 4 \leq S_D$ forces $S_D \geq 5$. Search proceeds to look for a solution. Note here that clause learning allows the solver to effectively skip from $S_D \leq 2$ to $S_D \leq 5$.

Figure 2.7: The relevant parts of the implication graph for the propagation of Example 2.3.6. Decision literals are shown double boxed, literals fixed through propagation are shown boxed, and relevant literals fixed through propagation of domain clauses are shown in dashed boxes. Dashed lines are used only to indicate the literals that resulted in the fail and the 1UIP is indicated with blue.



LCG and Schudule-or-Postpone are two different strategies that try to reduce the amount of search required within a CP framework. The Schedule-or-Postpone strategy is specific to scheduling problems yet is still a very general concept as it allows the search to determine dominance relations independently to the actual

constraints used to model the problem. This means that it can be used even in very complicated real-world models. A limitation of the Schedule-or-Postpone strategy is that in general it is incomplete. Weaker variants are typically used that still benefit from the dominance relations in finding feasible solutions. However these weaker version can still result in very large search trees and are thus less useful in proving optimality.

LCG complements CP with the nogood learning and automated search developed for SAT problems. An ongoing challenge and an interesting field of research is determining the most efficient ways to explain different global constraints to the SAT solver. A fundamental limitation of LCG however is the need to represent all elements of a variables domain as a boolean variable in the SAT solver. Although LCG allows for introducing constraints to a SAT solver, for scheduling problems with very large time-horizon, even enumerating the variables can be intensive.

In this section we have provided a brief introduction to CP and shown how CP can be complemented by a number of scheduling specific global constraints and search strategies. However modelling complex scheduling problems using this complementary approach can become challenging — typically several global constraints are connected through reified constraints. This can result in complex models that are hard to understand and hard to maintain. Furthermore it is difficult to develop efficient generic algorithms to solve such models as a lot of the structure of the problem, such as the AoN network, is lost when expressing scheduling problems using integer variables. In the next section we will introduce the scheduling-specific modelling language of CP Optimizer.

2.4 CP Optimizer

In this section we introduce relevant modelling concepts from CP Optimizer. While acknowledging that there are others CP solvers that specialise in solving scheduling problems, we consider CP Optimizer specifically as it currently provides a unique method for modelling and thus solving scheduling problems. Other solvers may have a different collection of constraint modelling constructs, yet for the thesis we are restricting ourselves to those available in traditional CP as presented in the previous section, and those available in CP Optimizer. The section is largely a synthesis of the relevant concepts from the recent paper by Laborie et al. (2018) as well as the CP Optimizer User Manual. However we point out that much of the style and syntax for presenting the variables and constraints is our own.

2.4.1 Background

In the early 1990s, ILOG began development of constraint-based approaches for scheduling problems, known as ILOG Scheduler. ILOG Scheduler was successfully applied to many industrial scheduling problems and successively grew to incorporate many different aspects of real-world scheduling problems. However there were a number of fundamentally limiting aspects to ILOG Scheduler. The growing complexity of the modelling language, combined with the lack of automatic search made using the tool cumbersome for users new to CP. Furthermore it had difficulty efficiently handling some important aspects of scheduling problems, such as the notion of different modes of execution. In 2007, the team behind ILOG Scheduler redesigned their CP tools and CP Optimizer. CP Optimizer puts a strong emphasis on a model-and-run development process, which would help align it with mathematical programming tools as advocated by Puget (2004), and introduces a minimal number of concepts to effectively model real-world scheduling problems. It also comes with a robust automatic search so that, like MIP, users can focus on the declarative model without having to develop overly complex search algorithms.

2.4.2 Decision Variables and Constraints

On top of the conventional integer variables in CP, CP Optimizer consider four other types of data structures for building models: `interval` variables, `sequence` variables, `state` functions, and `cumul` functions. By convention, the term *variable* is used to indicate that the data structure can be used in choice points during search. On the other hand, the term *function* is used to indicate that the data structure is only used to build more complicated constraints and is never used as a choice point in search. However, to ensure modelling consists of an objective function, a set of constraints, and a set of decision variables, we relax these semantics. Throughout this thesis we consider all of these different data structures to be *decision variables*. In this section we will introduce each of these types of decision variables and the constraints the can be defined on them.

Interval Variables

Overview: An interval variable, $\alpha \in$ `interval`, an interval of time not necessarily of a fixed length that we may or may not need to consider in our schedule. Each interval variable can be considered to be two integer variables representing the start

time $s(\alpha) \in \texttt{integer}$ and the end time $e(\alpha) \in \texttt{integer}$, and a boolean variable representing the presence status $x(\alpha) \in \texttt{boolean}$ [5]. Interval variables are *optional* in the sense that they are either *absent* ($x(\alpha) = 0$), or they are *present* ($x(\alpha) = 1$). If the interval variable must be present then we say it is *compulsory*. The domain of an interval variable $\texttt{dom}(\alpha)$ is a subset of $\{\perp\} \cup \{[s,e) | s \in dom(s(\alpha)), e \in dom(e(\alpha)), s \leq e\}$, where $\perp$ indicates the case where the interval is absent. Therefore when an interval variable is fixed it is either absent $\alpha = \perp$ or present with a fixed interval $\alpha = (s,e)$. In this case, s and e represent the *start* and *end* of the interval respectively and has *length* $l = e - s$.

Throughout the thesis, interval variables will be defined with respect to two parameters: (1) a boolean parameter indicating presence status (*cond*) and (2) a tuple to indicate the minimum and maximum lengths (l^{min}, l^{max}) of the interval. The presence condition is compulsory (*comp*), optional (*opt*), or a boolean expression dependent on fixed parameters. If the interval has a fixed length a single parameter will be given in place of the minimum and maximum length tuple. The convention for describing interval variables is as follows,

$$\alpha \in \texttt{interval}(cond, (l^{min}, l^{max})) \tag{2.8a}$$

Expression on Interval Variables: For each interval variable α, the expressions that provide access to the the start time integer variable $(s(\alpha))$, the end time integer variable $(e(\alpha))$, and the presence boolean variable $(x(\alpha))$, respectively, are indicated as follows,

$$\texttt{startOf}(\alpha, abs) \tag{2.8b}$$
$$\texttt{endOf}(\alpha, abs) \tag{2.8c}$$
$$\texttt{presenceOf}(\alpha) \tag{2.8d}$$

In constraints (2.8b) and (2.8c) the second parameter, *abs*, refers to the value that is returned if the interval variable is absent, i.e. $x(\alpha) = 0$. These expressions can be used to, in theory, model any constraints that one could typically write with respect to conventional integer variables. However, in our experience only

[5] Strictly speaking, in CP Optimizer, interval variables are atomic variables that cannot be decomposed. However, throughout this thesis it is adequate to consider interval variables as two integer variables and a boolean variables

the `presenceOf` expression is ever required as there are often more efficient methods for building constraints on the start and end times of interval variables then considering the corresponding integer variables explicitly. For example, next we will discuss how precedence constraints can be modelled such that information about the *temporal network* is captured.

Precedence Constraints / Temporal Network: Recall that in PS an AoN network is a network where activities are represented by nodes and the generalised precedence constraints are represented by arcs. A temporal network, also known as a *precedence graph*, is a generalisation of this concept where now both the start time and end time of the activities (intervals) are represented by nodes and again generalised precedence constraints are represented by arcs between the nodes.

The minimum and maximum lengths of an interval variable, as introduced when defining the variable (2.8a), are used when creating the temporal network. The minimum length of an interval variable is represented by a weighted arc from the start time node to the end time node with arc weight equal to the minimum length. Similarly, the maximum length of an interval variable is represented by a weighted arc from the end time node to the start time node with arc weight equal to the negative of the maximum length. The temporal network provides information to the solver about the structure of the problem and, like the AoN in the RCPSP/max, is used for a range of different purposes.

The following global constraints are used to build a temporal network in CP Optimizer,

$$\texttt{startBeforeStart}(\alpha_i, \alpha_j, \delta_{i,j}) \tag{2.8e}$$
$$\texttt{endBeforeStart}(\alpha_i, \alpha_j, \delta_{i,j}) \tag{2.8f}$$
$$\texttt{startBeforeEnd}(\alpha_i, \alpha_j, \delta_{i,j}) \tag{2.8g}$$
$$\texttt{endBeforeEnd}(\alpha_i, \alpha_j, \delta_{i,j}) \tag{2.8h}$$
$$\texttt{startAtStart}(\alpha_i, \alpha_j, \delta_{i,j}) \tag{2.8i}$$
$$\texttt{endAtStart}(\alpha_i, \alpha_j, \delta_{i,j}) \tag{2.8j}$$
$$\texttt{startAtEnd}(\alpha_i, \alpha_j, \delta_{i,j}) \tag{2.8k}$$
$$\texttt{endAtEnd}(\alpha_i, \alpha_j, \delta_{i,j}) \tag{2.8l}$$

Constraints (2.8e)-(2.8h) model generalised precedence constraints, where weighted

arcs are added from the start/end of one activity to the start/end of another activity with arc weight $\delta_{i,j}$. Constraints (2.8i)-(2.8l) provide a shorthand way of stating that the start/end of one activity must happen exactly a certain time before the start/end of another activity. With respect to the temporal network, an arc is added from the start/end of the first interval to the start/end of the second with arc weight $\delta_{i,j}$ and another arc is added from the second to the first with arc weight $-\delta_{i,j}$. Therefore, the advantage of expressing of expressing precedence constraints by using, for example `startBeforeStart` constraints instead of building the equivalent constraints using the `startOf` expressions, is that the solver can keep track of the temporal network.

Boolean Constraints / Logical Network: A *logical network* is also built which aggregates all the information regarding the interaction of the presence statuses between variables. More explicitly, all constraints between the presence statuses of the form

$$[\neg]\texttt{presenceOf}(\alpha_1) \wedge [\neg]\texttt{presenceOf}(\alpha_2)$$

are used to construct a boolean network similar to one proposed by Brafman (2001). The positive and negative literals of each boolean variable are represented as a node, and if a constraint exists such that one literal implies another then an arc from the first to the second is added to the network. The logical network allows for constant time access to logical relations that can be inferred between any two intervals. It also allows traversals to help understand which intervals imply the presence or absence of other intervals.

Constraints on groups of interval variables: There are a range of global constraints defined over groups of interval variables. The following are modelled as follows,

$$\texttt{span}(\alpha, \{\beta_1, ..., \beta_n\}) \tag{2.8m}$$

$$\texttt{alternative}(\alpha, \{\beta_1, ..., \beta_n\}) \tag{2.8n}$$

$$\texttt{isomorphism}(\{\alpha_1, ..., \alpha_m\}, \{\beta_1, ..., \beta_n\}) \tag{2.8o}$$

Informally, the `span` constraint ensures the α intervals spans the β intervals. The `alternative` constraint states that the β intervals are different alternatives of the α interval. The `isomorphism` constraint is used to unify two separate sets of intervals, here the α and β, which in fact are modelling the same set of intervals. More formally,

- the span constraint holds if and only if:

$$(x(\alpha) = 0) \iff \bigwedge_{i \in [1,n]} (x(\beta_i) = 0)), \text{ and}$$

$$(x(\alpha) = 1) \iff \begin{cases} (\bigvee_{i \in [1,n]} (x(\beta_i) = 1)) \\ (s(\alpha) = \min_{i \in [1,n): x(\beta_i) = 1} s(\beta_i)) \\ (e(\alpha) = \max_{i \in [1,n): x(\beta_i) = 1} e(\beta_i)) \end{cases}$$

- the alternative constraint holds if and only if:

$$(x(\alpha) = 0) \iff \bigwedge_{i \in [1,n]} (x(\beta_i) = 0)), \text{ and}$$

$$(x(\alpha) = 1) \iff \exists k \in [1,n] \begin{cases} (x(\beta_k) = 1) \\ \bigwedge_{i \in [1,n] \setminus k} (x(\beta_i) = 0) \\ s(\alpha) = s(\beta_k) \\ e(\alpha) = e(\beta_k) \end{cases}$$

- the isomorphism constraint holds if and only if:

$$\exists f : P(A) \to P(B), \quad \text{where} f \text{ is bijective, and}$$
$$\forall \alpha \in P(A) \begin{cases} s(\alpha) = s(f(\alpha)) \\ e(\alpha) = e(f(\alpha)) \end{cases}$$

  Where $P(A)$ is the set of present α interval variables and $P(B)$ is the set of present β interval variables.

Sequence Variables

Overview: A sequence variable, ρ, is a decision variable that is defined with respect to a set of interval variables A. A value of the sequence variable is a *permutation* $\pi : A \to \{\perp\} \cup \{1, ..., |A|\}$, which is a mapping from the interval variables to either the symbol $\perp$ stating the interval is absent, or a positive integer bounded by the number of interval variables being considered. Sequence variables inherently constraint a permutation π such that (1) each interval variable $\alpha \in A$ is absent $x(\alpha) = 0$ if and only if $\pi(\alpha) = \perp$; (2) if an activity is present $(x(\alpha) = 1)$ it is mapped to an integer equal to or less than the number of present interval variables;

and (3) for every pair of interval variables, if they are both present then they cannot have the same position. Hence the domain of a sequence variable consists of all the permutations of the present interval variables of A. Furthermore, each interval $\alpha \in A$ is also associated with a non-negative integer parameter known as a *type*, denoted by $T(\alpha)$.

In the thesis, sequence variables are defined either *explicitly* by defining all of the interval variables and interval types up front as follows,

$$\rho \in \texttt{sequence}((\alpha, T(\alpha))_{\forall \alpha \in A}) \tag{2.9a}$$

Or *constructively* by first defining a sequence variable and then stating which interval variables, along with their types, are considered by the sequence variable, as follows,

$$\rho \in \texttt{sequence} \tag{2.9b}$$

$$\texttt{inSequence}(\rho, \alpha, T(\alpha)) \quad \forall \alpha \in A \tag{2.9c}$$

The sequence variable on their own do not enforce any constraints on the interval start and end times. An interval α could be sequenced before an interval variable α' in a sequence ρ without any impact on the relative position between the start and end times of α and α'.

NoOverlap Constraint: To ensure that the interval variables considered by a sequence variable do not overlap, the `noOverlap` constraint is defined as follows,

$$\texttt{noOverlap}(\rho, M) \tag{2.9d}$$

Where M is a transition distance matrix that specifies the minimal distance that must separate two consecutive intervals in the sequence depending on their type. The number of rows and columns of M is equal to the number of different types associated with the interval variables in the sequence. The transition matrix does not necessarily need to satisfy the triangle inequality[6]. More formally, the

[6]CP Optimizer contains two versions of the noOverlap constraint depending on whether or not the transition matrix applies between all predecessors/successors or only between immediate ones. Here we present the more general version of the constraint but note that in the case where M

NoOverlap constraint ensures that for every two unique interval variables $\alpha, \alpha' \in A$, if they are both present then

$$\pi(\alpha) < \pi(\alpha') \Leftrightarrow e(\alpha) + M[T(\rho, \alpha), T(\rho, \alpha')] \leq s(\alpha')$$

Relative Position Constraints: To constrain certain interval variables to certain positions in the sequence, the following global constraints are defined,

$$\texttt{first}(\rho, \alpha) \tag{2.9e}$$
$$\texttt{last}(\rho, \alpha) \tag{2.9f}$$
$$\texttt{before}(\rho, \alpha, \alpha') \tag{2.9g}$$
$$\texttt{prev}(\rho, \alpha, \alpha') \tag{2.9h}$$

The `first` constraint ensures that if α is present then $\pi(\alpha) = 1$. The `last` constraint ensures that if α is present then $\pi(\alpha) = |\{\alpha \in A : x(\alpha) = 1\}|$. The `before` constraint ensures that if two unique interval variables $\alpha, \alpha' \in A$ are both present then $\pi(\alpha) < \pi(\alpha')$. The `prev` constraint is a stronger version of this constraint where the first interval must directly precede the second, i.e., $\pi(\alpha) = \pi(\alpha') - 1$.

State Variables

Motivation: Many scheduling problems also involve reasoning about the state of certain resources. Typical examples include the temperature of an oven in which jobs can only be executed within certain ranges, the type of material that is present in a tank, or the current tool installed on a machine. Jobs can only be completed when resources are in certain states and it might require some time to transition between different states of the resouce. To account for such situations, CP Optimizer introduces *state functions*.

Overview: A *state function*, ψ, is a decision variable that represents a set of non-overlapping intervals of time that we refer to as segments. We refer to the intervals of a state function as segments to distinguish them from interval variables. Each of the segments i are associated with a non-negative integer value v_i that represents the state of the function over the segment. Hence a value of state function ψ is

satisfies the triangle inequality, these two versions are equivalent.

denoted as $([s_i, e_i) : v_i)_{i\in[1,n]}$, where s_i and e_i are the start and end times of each segment, respectively. Inherently state functions constrain $s_i < e_i$ for each segment. For a fixed state function ψ the set of points where the state function is associated with a state is defined as $D(\psi) = \bigcup_{i\in[1,n]}[s_i, e_i)$. For a point $t \in D(\psi)$, the unique segment of the function that contains t is denoted by $[s(\psi, t), e(\psi, t))$ and the value of the segment is denoted by $\psi(t)$.

Like sequence variables, state functions are defined with respect to a set of interval variables. However, the interactions between the interval variables and state function can be somewhat more complicated to those with respect to sequence variables. For that reason, state functions are only defined constructively as follows,

$$\psi \in \texttt{state}(M) \tag{2.10a}$$

Where M is again a transition distance matrix that represents the minimum distance that must separate two consecutive states in the state function. The dimension of M is equal to the maximum possible value that a segment might take.

Constraints: There exist a range of constraints to define the different relations between interval variables and state functions. A number of constraints that are used in this thesis are expressed as follows,

$$\texttt{alwaysNoState}(\psi, \alpha) \tag{2.10b}$$
$$\texttt{alwaysConstant}(\psi, \alpha, \leftrightarrow) \tag{2.10c}$$
$$\texttt{alwaysEqual}(\psi, \alpha, v, \leftrightarrow) \tag{2.10d}$$

The `alwaysNoState` constraint ensures that the state is not defined over the interval of the interval variable, i.e., $D(\psi) \cap [s(\alpha), e(\alpha)) = \emptyset$. For the `alwaysConstant` constraint, whenever α is present, state function ψ must be defined everywhere between the start and end of interval α and be constant over this interval, i.e., $[s(\alpha), e(\alpha)) \subseteq [s(\psi, s(\alpha)), e(\psi, s(\alpha))$. The arrow parameter $\leftrightarrow$ is a shorthand representation for start and end alignments, and may be either $-$ (no alignment), $\leftarrow$ (start alignment only), $\rightarrow$ (end alignment only), or $\leftrightarrow$ (both start and end alignment). If start alignment is required then the start of the interval variable must occur at the start of the corresponding interval from the state function, i.e., $s(\alpha) =$

$s(\psi, s(\alpha))$. Similarly, if end alignment is required then the end of the interval variable must occur at the end of the corresponding interval from the state function, i.e., $e(\alpha) = e(\psi, e(\alpha))$. The `alwaysEqual` constraint is equivalent to an `alwaysConstant` constraint with the addition that the corresponding interval from the state function must equal a specific value, i.e., $f(s(\alpha)) = v$.

Cumul Function Expressions

Motivation: Many scheduling problems consider cumulative resources, where the cumulative usage of the resource is represented as a function over time. For the renewable resources in the RCPSP(/max), activities increase the cumulative resource usage function at their start times and decrease it when they release the resource at their end time. However in many real-world scheduling problems there exist activities that can produce and consume resources. Take for example the contents of a tank or activities that produce and consume money. For these cases, the resource level can also be described as a function of time. Hence the cumulated contribution of certain intervals on the resource can be represented by a function of time, and then constraints can be posted on this function.

Overview: A *cumul function*, f, is function that represents the sum of individual contributions of intervals to the total resource usage. These individual contributions are known as *elementary function expressions* and describe the contribution from a single interval variable or a fixed interval of time to the resource usage. Let $f(t)$ be the value of cumul function f at time t. Note that it is typically inefficient to store the value of at each time point, instead cumul functions use data structures where the value at each point in time can be inferred based on the time points that change the value, i.e., the starts and ends of intervals. Interval variables can interact with cumul functions, and thus we define cumul functions constructively as follows,

$$f \in \texttt{cumul} \tag{2.11a}$$
$$f \mathrel{+}= \texttt{pulse}(\alpha, h) \tag{2.11b}$$
$$f \mathrel{+}= \texttt{stepAtStart}(\alpha, h) \tag{2.11c}$$
$$f \mathrel{+}= \texttt{stepAtEnd}(\alpha, h) \tag{2.11d}$$
$$f \mathrel{-}= \texttt{pulse}(\alpha, h) \tag{2.11e}$$
$$f \mathrel{-}= \texttt{stepAtStart}(\alpha, h) \tag{2.11f}$$
$$f \mathrel{-}= \texttt{stepAtEnd}(\alpha, h) \tag{2.11g}$$

When a cumul function f is first defined (2.11a), the value at each time point is zero, i.e., $f(t) = 0$ for all t. Constraints (2.11b)-(2.11g) demonstrate how interval variables can affect the value of a cumul function. The `pulse` constraint ensures that if the interval variable is present, the value of the cumul function increases (or decreases) by positive integer h for the duration of the interval, i.e., $f(t)\ +\!= h$ (or $f(t)\ -\!= h$) for all $t \in [s(\alpha), e(\alpha))$. Likewise, the `stepAtStart` and `stepAtEnd` constraints increase (or decrease) the cumul function by positive integer h for all time points after the start and end of the interval, i.e., $f(t)\ +\!= h$ (or $f(t)\ -\!= h$) for all $t \in [s(\alpha), \infty))$ or $t \in [e(\alpha), \infty))$, respectively. For all the `pulse stepAtStart` and `stepAtEnd` constraints it is possible to add a fixed range $[s, e)$ instead of an interval variable.

Constraints: Constraints exist to restrict the possible values that a cumul function can take at different time points. The types considered in this thesis are as follows,

$$h^{min} \leq f \leq h^{max} \tag{2.12a}$$

$$\texttt{alwaysIn}(f, \alpha, h_{min}, h_{max}) \tag{2.12b}$$

Constraints (2.12a) ensure that the value of the cumul function never is less than h^{min} or more than h^{max} at any time point, i.e., $h^{min} \leq f(t) \leq h^{max}$ for all t. The `alwaysIn` constraint (2.12b) is a relaxation of this, where the values of the cumul function are again restricted to between h^{min} and h^{max} but only during the interval value α if α is present, i.e., $h^{min} \leq f(t) \leq h^{max}$ for all $t \in [s(\alpha), e(\alpha))$ if $x(\alpha) = 1$.

2.4.3 Modelling the RCPSP/max

For continuity, we describe how the reference problem, the RCPSP/max, is typically modelled in CP Optimizer. Each activity $i \in V$ is assigned a compulsory interval variable α_i, with a fixed duration d_i. A typical CP Optimizer model for the RCPSP/max is then,

$$\min \quad \max_{i \in V}(\texttt{endOf}(\alpha_i)) \tag{2.13a}$$

$$\text{s.t.} \quad \texttt{startBeforeStart}(\alpha_i, \alpha_j, \delta_{i,j}) \qquad \forall (i,j) \in A \tag{2.13b}$$

$$f_k\ +\!= \texttt{pulse}(\alpha_i, r_{i,k}) \qquad \forall k \in R; i \in V \tag{2.13c}$$

$$f_k \leq R_k \qquad \forall k \in R \qquad (2.13d)$$

$$\alpha_i \in \texttt{interval}(comp, d_i) \qquad \forall i \in V \qquad (2.13e)$$

$$f_k \in \texttt{cumul} \qquad \forall k \in R \qquad (2.13f)$$

The objective function (2.13a) is to minimise the maximum completion time of any of the activities. Constraints (2.13b) enforce the generalised precedence constraints. Each resource is represented by a single cumul function expression (2.13f) that represents the sum of the individual resource requirements during the interval variables (2.13c) with a constant capacity over the considered project (2.13d). For each activity an interval variable is defined with length equal to the duration of the activity and present status fixed such that it must be present (2.13e). Assuming that the number of precedence constraints is proportional to the number of activities, this model is linear with respect to both the number of variables and constraints.

2.4.4 Search and Constraint Propagation

CP Optimizer has a very powerful default search that incorporates many of the ideas from the serial schedule generation schemes, constraint propagation, and the Schedule-Or-Postpone strategies discussed in the previous sections. The automated search has three distinct phases: initial heuristics are used to generate feasible solutions; an improvement phase tries to improve the feasible solution; and finally the solver focusses on determining an optimality proof. For a full review of the default search we refer the reader to the following references (Laborie et al., 2018; Godard et al., 2005; Vilim et al., 2015)

Initial solutions are generated using a range of heuristics that utilise the temporal network. A topological sort is computed over the temporal network to determine information that will help determine feasible schedules through variable orderings. An example of the topological sort is, as already discussed in section 2.1.2, to find the strongly connected components. The methods for building schedules depends on the type of objective function being considered by the problem.

Feasible solutions are improved using a self-adapting Large Neighbourhood Search (SA-LNS). LNS (Shaw, 1998) is a well-known, solution-based metaheuristic for the systematic search for sufficiently good solutions to an optimization problem. The process is based on the continual relaxation and re-optimization of a feasible solution. A significant challenge of implementing LNS on scheduling problems is that typically most commonly used heuristics generate a schedule with fixed

start times, in which relaxing a solution by unfreezing the start time of a subset of activities in the schedule provides limited flexibility to re-optimize the relaxed solution. To provide the framework more flexibility, Godard et al. (2005) relax initial solutions into partial order schedules (POSs) (Policella et al., 2004), after which a LNS relaxation is applied with respect to the POS. The optimization component of CP Optimizer's default search is a SA-LNS, for which the fundamental mechanics are describe in Godard et al. (2005). Relaxations of the problem are reconstructed using different variants of the SetTimes algorithm as discussed in Section 2.3.4.

Once the solver decides it has becomes difficult to improve the solution, *Failure Directed Search* (FDS) is used to efficiently close the search tree. FDS was first described by (Vilim et al., 2015) and operates on the assumption that the current problem is infeasible. Therefore FDS directs the search into conflicts in order to prove that the current branch is infeasible. Choices that fail the most are preferred. When FDS is being used, stronger filtering algorithms are used. For example, typically time-tabling is used to enforce constraints over cumul expressions, however when FDS begins, the time-table edge finding filtering algorithm is used.

As will be used extensively in this thesis, there are applications where there exist a group of key decision variables, such that once these variables are fixed, it is straightforward to extend the partial solution to the remaining variables. A *search phase* is a group of decision variables of the same type (integer, interval or sequence). Search phases can be used to instruct the solver to consider groups of variables in a specified order: the decision variables of the first phase are instantiated before the variables in the second phase and so on.

2.4.5 Constraint Propagation

CP Optimizer uses a range of filtering algorithms to enforce the constraints of the problem. Details regarding all of the different filtering algorithms are not necessarily made public and as the emphasis of this thesis is on the modelling, complete knowledge of all the different filtering algorithms is not necessary. However as different filtering algorithms will be used depending on how a problem is modelled, it is worthwhile to consider the following:

- Exploiting the temporal network - As all precedence constraints are aggregated into the temporal network, it is possible to perform filtering on all temporal constraints at once extremely efficiently using improved versions of shortest path algorithms such as the Bellman-Ford algorithm. Therefore it is generally beneficial to provide the temporal network with as much infor-

mation as possible.

- Exploiting the logical network - The propagation of precedence constraints on the temporal network exploit the implication relations between presence intervals in the logical network. These propagations are well summarised in Laborie and Rogerie (2008) and Laborie et al. (2009). Hence the presence status of an interval variable does not necessarily have to be fixed in order for its precedences to be considered while propagating the temporal network.
- Timetabling - The default filtering algorithm for all sequence variables, state functions and cumul functions are modified versions of the *timetable* filtering algorithm discussed in Section 2.3.2. This is discussed in greater detail in Chapter 5 of this thesis.
- Edge Finding - If a cumul function consists of only positive pulses (2.11b) and the only constraints on the cumulative function is an upper limit (2.12a), then the *edgefinder* cumulative constraint can also be used. In Chapter 4, we will see how this can help improve performance.

CHAPTER 3

Symmetry Breaking in the High-Multiplicity RCPSP/max

3.1 Introduction

With respect to job shop scheduling, Hochbaum and Shamir (1991) describe high-multiplicity scheduling problems as problems that consist of many jobs which can be partitioned into relatively few groups, where all the jobs within each group are identical. Typically these groups of identical jobs are referred to as *classes* (Van Der Veen and Zhang, 1996).

In this chapter we consider the high-multiplicity version of the RCPSP/max. With respect to our real-world problem, scheduling the finite resources of the automated system to process multiple tests in parallel in the least amount of time can be modelled as the high-multiplicity RCPSP/max. Each protocol is represented by a project class. Tests with the same protocol are represented as projects from the same class. The activities of a project represent the activities that must be completed by the system, and a set of generalised precedence relations account for the timings of the chemical processes. The resources of the automated system are modelled as renewable resources.

In our conference paper Edwards et al. (2017), we claimed that there exists symmetry between tests of the same protocol and that this symmetry can be bro-

ken by considering an additional set of precedence constraints. Furthermore we claimed that these additional precedence constraints allowed CP Optimizer to, on average, find better solutions on a real-world data set compared to the existing approach. This chapter presents the work in (Edwards et al., 2019), which extends (Edwards et al., 2017) by making the following additional contributions:

- Model the problem as the high-multiplicity version of the well-known RCPSP/max, and contextualise this problem within the scheduling literature;
- Formally prove the existence of the symmetry between activities of the same index from projects of the same class;
- Consider two different symmetry breaking approaches: (1) by reformulating the model to reduce the number of decision variables, and (2) by introducing additional precedence constraints to break symmetries;
- Propose two mixed-integer programming (MIP) formulations based on integer decision variables that eliminate the symmetry between identical projects;
- Complete a computational study to analyse the different approaches of symmetry breaking in various MIP and CP-based scheduling models; and
- Demonstrate the efficiency of symmetry breaking on relevant benchmark problems in the multi-project scheduling problem library (MPSPlib).

The structure of the chapter is as follows. Section 3.2 gives the problem description of the high-multiplicity RCPSP/max and then contextualises the problem within the literature. Section 3.3 provides a formal proof of the symmetry between projects of the same class. Section 3.4 adapts a well known discrete-time MIP model for the classical resource-constrained project scheduling problem (RCPSP) to the high-multiplicity RCPSP/max and proposes two MIP models based on integer variables that reduce the number of variables required. Section 3.5 includes how the symmetry can be broken for two well known models based on CP. Section 3.6 includes two computational experiments and a discussion of the results. The chapter concludes by discussing future research directions.

3.2 Background

3.2.1 Problem description

An instance of the high-multiplicity RCPSP/max is represented by a single, master AoN, $\mathcal{N} = (V, A)$. In the network, activities are represented by vertices and

generalised precedence constraints are represented by weighted arcs.

Individual projects, which can be divided into a set of project classes C, are considered. The class defines the set of n_c activities that must be processed in order to complete a single project of class $c \in C$. The set of activities for project class $c \in C$ is defined $V_c := \{0, 1, ..., n_c, n_c + 1\}$, where 0 and $n_c + 1$ are dummy start and end activities for projects of that class. The class also defines the set of precedence relations between these activities represented by the set A_c. Here $(i, i') \in A_c$ represents a precedence from activity $i \in V_c$ to activity $i' \in V_c$ for all projects $p \in P$ of class $c \in C$. For each project class $c \in C$, a number of multiples of that project, m_c, must be completed. Let $P_c := \{0, ..., m_c - 1\}$ denote the projects of class $c \in C$.

For notational consistency we introduce a dummy class, $0 \in C$. Class zero contains a single dummy project $m_0 = 1$, with two dummy activities $(0, 0, 0)$ and $(\omega, 0, 0)$ that can be interpreted as the global start and end activities respectively. For brevity we refer to the dummy end node as simply ω where possible. The global node set is thus defined as $V := \{(i, p, c) | i \in V_c, p \in P_c, c \in C\}$. The global arc set is the union of three sets; (1) the project arcs defined by the classes, $A^{real} := \{((i, p, c), (i', p, c)) | (i, i') \in A_c, p \in P_c, c \in C\}$, (2) the arcs from the global dummy start activity to the project dummy starts, $A^{start} := \{(0, 0, 0), (0, p, c) | p \in P, c \in C \setminus \{0\}\}$ and (3) the arcs from the project dummy ends to the global dummy end, $A^{end} := \{(n_c + 1, p, c), (\omega, 0, 0) | p \in P, c \in C \setminus \{0\}\}$. Hence $A := A^{real} \cup A^{start} \cup A^{end}$.

A *schedule* is an assignment of start times to activities, $\mathcal{S} := \{S_{i,p,c} | (i, p, c) \in \mathcal{V}\}$, where $S_{i,p,c}$ denotes the start time of activity $(i, p, c) \in \mathcal{V}$. Two distinct sets of constraints are considered:

1. *Generalised Precedence Constraints*: each precedence $((i, p, c), (i', p', c')) \in A$ has an associated time lag $\delta_{(i,p,c),(i',p',c')}$, which states that activity (i, p, c) starts at least $\delta_{(i,p,c),(i',p',c')}$ time-units before activity (i', p', c'), i.e.,

$$S_{i',p',c'} - S_{i,p,c} \geq \delta_{(i,p,c),(i',p',c')}, \quad \forall ((i, p, c), (i', p', c')) \in \mathcal{A} \tag{3.1}$$

   If a time lag is non-negative it is referred to as a *minimum time lag*. If a time lag is negative it is referred to as a *maximum time lag*. A schedule $\mathcal{S}$ is *time-feasible* if all of the time lags are respected, i.e., (3.1) holds.

2. *Resource Constraints*: we are given a set of renewable resources $\mathcal{R}$. Each resource $k \in \mathcal{R}$ has a capacity of R_k units. In order to be processed, activity

$i \in V_c$ of class $c \in C$, requires $r_{i,c,k}$ units of resource $k \in \mathcal{R}$ for the entire duration, $d_{i,c}$, of the activity. At each time t over some horizon H, the demand for resource $k \in \mathcal{R}$, denoted by $r_k(\mathcal{S}, t)$, must not exceed the capacity R_k, i.e.,

$$r_k(\mathcal{S}, t) = \sum_{(i,p,c) \in \mathcal{V}: t \in [S_{i,p,c}, S_{i,p,c} + d_{i,c}[} r_{i,c,k} \leq R_k, \quad \forall\, t \in H, k \in \mathcal{R} \tag{3.2}$$

A schedule $\mathcal{S}$ is *resource-feasible* if the demand for a resource never exceeds the capacity of the resource, i.e., (3.2) holds. A schedule is *feasible* if the schedule is both time-feasible and resource-feasible. The objective is to find the schedule with the lowest makespan, i.e., minimise S_ω. For completion it is noted that the duration of all dummy activities is zero. The resource requirements of all dummy activities for all resources are zero. The time lags associated with each dummy arc, i.e., those in A^{start} and A^{end}, are zero.

3.2.2 Literature review

To the best of our knowledge the high-multiplicity RCPSP/max has not been considered in the literature. However there exists many closely related well-studied problems. Here we contextualise the problem considered in this chapter within the literature.

As claimed by Kolisch (2015), the RCPSP is probably the most studied problem in project scheduling. The RCPSP has been extremely well studied in the scheduling literature, due to a combination of being easy to state, relevant in practice and yet hard to solve; famously being proven NP-Complete by Blazewicz et al. (1983). The RCPSP is a special case of the RCPSP-max where all the precedence constraints have a time lag equal to the duration of the predecessor activity. These precedence constraints are referred to as *simple precedence constraints* and are more typically presented as a zero time lag between the end of the predecessor activity and the start of the successor activity. A vast number of variations to the RCPSP have been considered in the literature. For a detailed overview of common problem extensions, we refer the reader to Schwindt and Zimmermann (2015a), Schwindt and Zimmermann (2015b) and Hartmann and Briskorn (2010).

As the RCPSP/max is a generalisation of the RCPSP, finding an optimal solution remains in general NP-hard. Moreover as shown by Bartusch et al. (1988) even proving that a problem is feasible given an unlimited time horizon is NP-Complete. Finally with respect to the naming convention of Brucker et al. (1999a),

the RCPSP/max is denoted $PS|temp|C_{max}$. When pre-emption is not allowed, which is the case considered in this chapter, it is possible to represent start/end, end/end and end/start generalised precedence constraints as start/start precedences, through the so-called Bartusch et al. transformations (Bartusch et al., 1988).

The state-of-the-art solution technique to the RCPSP/max (Schutt et al., 2013c) is a constraint programming approach that includes SAT-inspired nogood learning based on lazy-clause generation (Ohrimenko et al., 2009) and conflict-driven search. The approach closed 573 out of the 631 open problems in well established benchmarks and improved a further 51 upper bounds of the 58 remaining open problems. We refer to CP solvers that use lazy-clause generation as *learning CP solvers*.

Learning CP solvers have also been used on a number of generalisations of the RCPSP/max considered in the literature. Schutt and Stuckey (2016) explored different types of learning languages to explain producer and consumer types of resource constraints, i.e., resources for which an activity can produce or consume at the start or end of the activity. The performance achieved is comparable with the best of the nine methods considered by Laborie (2003a). A learning CP solver was successfully implemented by Kreter et al. (2017) on the more general RCPSP/max where calendar constraints are also considered, denoted RCPSP/max-cal. The authors develop a specialised propagator for the cumulative resource constraints taking the calendar constraints into account. Finally learning CP solvers has been successfully implemented in other related scheduling problems such as the classical RCPSP Schutt et al. (2011) and the flexible job shop scheduling problem Schutt et al. (2013a).

CP Optimizer been shown to perform well on a number of scheduling problems (Laborie et al., 2018). The default search incorporates a range of heuristics to find and improve feasible solutions. Vilim et al. (2015) show how this default search can be complimented by running in parallel a destructive lower bound method, which they name *failure-directed search*, after a sufficiently good solution has been found. This method was used to close many open instances from a range of different project scheduling problems, of which the RCPSP/max was one. For some problems the approach made remarkable improvements, for example in the multi-mode RCPSP the approach closes 535 out of the 552 open instances. However for the RCPSP/max the approach was only able to close one of the 58 instances left open by Schutt et al. (2013c).

The basic multi-project scheduling problem (BMPSP) is a special case of the

RCPSP where two or more projects which require the same renewable resources are scheduled in parallel. A common modelling approach in multi-project scheduling problems is to merge all projects into an artificial super-project with dummy start and end activities, as is done in this chapter. In some versions of the BMPSP a distinction is drawn between local and global resources (Confessore et al., 2007). A *local resource* is a resource that is only shared amongst activities from that project, whereas a *global resource* is shared by all activities from all projects. If a problem only considers local resources, then the BMPSP can be decomposed into separate instances of the RCPSP. In this chapter all resources are global, as this is the case in our motivating problem.

Neumann and Zhan (1995) point out that an instance of the RCPSP/max is feasible if and only if each maximal strongly connected component of the project network, which they refer to as a *cycle*, has a feasible schedule. This decomposition has been used frequently in particular for schedule generation schemes for the RCPSP/max, which are well summarised in Franck et al. (2001).

For a number of high-multiplicity scheduling problems, good-quality, polynomial-time approximation algorithms have been presented. These approaches generally derive dispatching rules from the solutions of the linear relaxation of scheduling problems and prove that the quality of these solutions are guaranteed to be within a certain bound of the optimal solution. Some examples of such algorithms for the minimum makespan high-multiplicity jobshop problem (Boudoukh et al., 2001; Masin and Raviv, 2014) and the 3-stage flowshop scheduling problem with identical jobs and uniform parallel machines (Verma and Dessouky, 1999).

3.3 Symmetry breaking

Symmetry breaking is a method that is widely used in optimization to reduce the search space. Excellent overviews of symmetry in CP and MIP are given by Gent et al. (2006) and Margot (2010) respectively. A symmetry is a mapping between variable-value pairs that maps feasible solutions to other feasible solutions with the same objective value. Two solutions are *symmetric* if there exists a symmetry that maps one solution to the other. The aim of symmetry breaking is to ideally eliminate all but one of the symmetric solutions.

In this section we prove the existence of symmetry between activities of the same index of projects from the same class. To provide some intuition to the notion of symmetry under investigation consider the following small example. Figure

3.1 presents the activity-on-node (AoN) network of the non-dummy activities of three projects from the same class. The node index (i, p, c) stands for activity $i \in \{1, 2, ..., 5\}$ from project $p \in \{1, 2, 3\}$ from class $c = 1$. For clarity arc values are omitted from the network, however we assume for feasibility that the projects contain no positive cycles. We will prove that the symmetry in this section can be broken by considering the additional precedence constraints, represented by the dotted arcs.

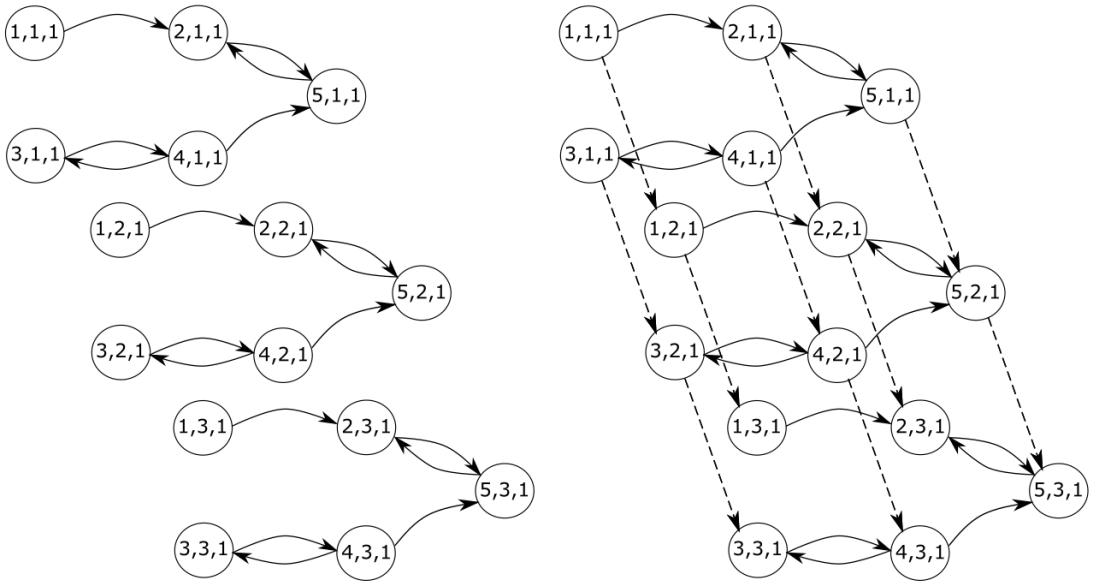


Figure 3.1: Intuition of the symmetry breaking constraints. Additional precedence constraints (dotted) can be added between activities of the same index between projects from the same class. Alternatively the symmetry can be removed through remodelling.

Kovács and Váncza (2006) prove the existence of symmetry between related activities of special types of sets of activities for the RCPSP, which they refer to as progressive pairs, and show how additional precedence constraints can be inferred between these activities under certain conditions. They then show how the symmetry breaking constraints hold for more general versions of the RCPSP, including those with non-negative time lags. As Kovács and Váncza (2010) do not explicitly consider negative time lags, the project networks are all direct acyclic graphs and hence do not contain any cycles. In this chapter we prove that the symmetry holds in the face of maximum time lags for identical projects, and thus for precedence graphs that contain cycles. Furthermore the proof by Kovács and Váncza (2010) is based on consecutively considering sets of identical activities in pairs. Our proof considers all projects of the same class simultaneously and shows how a certain schedules can be obtained directly from any feasible schedule, which is a novel

contribution in itself.

Our proof is based on swapping start times between activities in feasible schedules. We first consider any feasible schedule to any instance of the RCPSP/max and then show how a unique type of schedule can be determined by swapping activity start times in a specific way. To perform these swaps, we introduce the *start time ordering function*, $O^q_{i,c}(\mathcal{S})$, which denotes the qth smallest start time of activity index $i \in V_c$ amongst all the projects of class $c \in C$ according to schedule $\mathcal{S}$.

Thus given any schedule $\mathcal{S}$ a permutation of the schedule $\mathcal{S}^\pi$ can be defined by assigning the start times using the start time ordering function as follows,

$$\mathcal{S}^\pi_{i,p,c} := O^p_{i,c}(\mathcal{S}) \quad \forall\, i \in V_c, p \in P_c, c \in C \tag{3.3}$$

Hence for a given class $c \in C$ and activity index $i \in V_c$ the start time for project $p \in P_c$ is assigned the pth lowest start time of the corresponding activities in the original schedule $\mathcal{S}$. We will refer to the schedules $\mathcal{S}^\pi$ obtained by permuting the start times of a feasible schedule $\mathcal{S}$ using the start time ordering function as *permuted schedules*. We say $\mathcal{S}^\pi$ is the permuted schedule of $\mathcal{S}$.

A small example that demonstrates how the start time ordering function is used to reorder the start times of activities of the same index from projects of the same class is illustrated in Figure 3.2. The network associated with some precedence $(i, i') \in A_c$ for a class with $m_c = 5$, is shown on the left. Two schedules are shown on the right hand side. A rectangle is used to indicate the interval of the activity according to the corresponding schedules. The horizontal line indicates the project, on each line the rectangle on the left refers to activity index i, whereas the rectangle on the right refers to i'. The top schedule is some feasible schedule $\mathcal{S}$, the bottom schedule $\mathcal{S}^\pi$ is the permuted schedule of $\mathcal{S}$. The activities in $\mathcal{S}^\pi$ are ordered by the starting times in $\mathcal{S}$.

The following two lemmas show that a permuted schedule of any feasible schedule are resource-feasible and time-feasible, respectively.

Lemma 3.3.1. *For any feasible schedule $\mathcal{S}$, the permuted schedule $\mathcal{S}^\pi$ is resource-feasible*

Proof. As $\mathcal{S}$ is feasible, we know that (3.2) holds and hence $r_k(\mathcal{S}, t) \leq R_k$ for all $t \in H$ and $k \in \mathcal{R}$. As activities of the same index of projects from the same class have the same duration $d_{i,c}$ and resource requirements $r_{i,c,k}$, any reordering of start times will not change the resource profile for all resources, and hence $r(\mathcal{S}^\pi, t) = r(\mathcal{S}, t) \leq R_k$. □

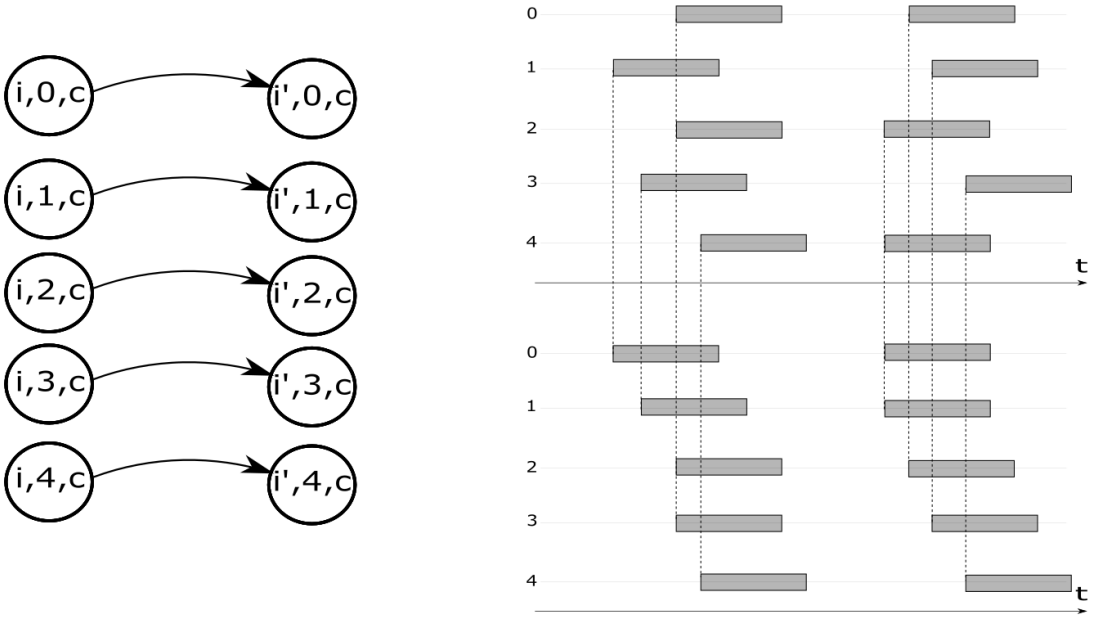


Figure 3.2: A small example that demonstrates how the start time ordering function is used to construct unique schedules. The figure on the left is the subgraph of an AoN for activities from five different projects from the same class with the same activity index. On the top right is a gantt chart of a feasible schedule $\mathcal{S}$. The figure on below shows activities in the permuted schedule $\mathcal{S}^\pi$ ordered by the start times in $\mathcal{S}$.

Lemma 3.3.2. *For any feasible schedule $\mathcal{S}$, the permuted schedule $\mathcal{S}^\pi$ is time-feasible*

Proof. As we know $\mathcal{S}$ is feasible, it is also time-feasible and thus (3.1) holds. Hence

$$\min_{p \in P_c}(S_{i',p,c} - S_{i,p,c}) \geq \delta_{(i,p,c),(i',p,c)} \quad \forall\ (i,i') \in A_c,\ c \in C.$$

The dummy precedences associated with arc sets A^{start} and A^{end} trivially hold in $\mathcal{S}^\pi$. To complete the proof it is sufficient to instead prove that

$$S^\pi_{i',p,c} - S^\pi_{i,p,c} \geq \min_{p' \in P_c}(S_{i',p',c} - S_{i,p',c}) \quad \forall\ (i,i') \in V_c,\ p \in P_c,\ c \in C.$$

For the sake of contradiction assume that there exists some pair of activities, $i, i' \in V_c$, from some project $\hat{p} \in P_c$ in class $\hat{c} \in C$ such that the time lag from activity $(i,\hat{p},\hat{c})$ to activity $(i',\hat{p},\hat{c})$ is strictly less than all corresponding time lags in the initial schedule, i.e. $S^\pi_{i',\hat{p},c} - S^\pi_{i,\hat{p},c} < \min_{p \in P_c}(S_{i',p,c} - S_{i,p,c})$. This assumption implies that in the original schedule, there does not exist any project $p \in P_{\hat{c}}$ from class $\hat{c} \in C$, where $S_{i,p,\hat{c}} \geq S^\pi_{i,\hat{p},\hat{c}}$ and $S_{i',p,\hat{c}} \leq S^\pi_{i',\hat{p},\hat{c}}$.

We now consider two sets of projects of class $\hat{c}$. Let A be the set of projects

in $P_{\hat{c}}$ for which the starting time of activity index i in the original schedule is strictly less than the starting time of activity index i of the violating project in the permuted schedule, i.e. $A := \{p \in P_{\hat{c}} : S_{i,p,\hat{c}} < S^{\pi}_{i,\hat{p},\hat{c}}\}$. Let B be the set of projects for which the starting time of activity i' in the original schedule is less than or equal to the starting time of activity i' of the project $\hat{p}$ in the permuted schedule, i.e. $B := \{p \in P_{\hat{c}} : S_{i',p,\hat{c}} \leq S^{\pi}_{i',\hat{p},\hat{c}}\}$.

As we know there does not exist any project $p \in P_c$ such that, $S_{i,p,\hat{c}} \geq S^{\pi}_{i,\hat{p},\hat{c}}$ and $S_{i',p,\hat{c}} \leq S^{\pi}_{i',\hat{p},\hat{c}}$, hence for all $p \in P_c$, if $S_{i',p,\hat{c}} \leq S^{\pi}_{i',\hat{p},\hat{c}}$ then $S_{i,p,\hat{c}} < S^{\pi}_{i,\hat{p},\hat{c}}$. Equivalently this implies that $B \subseteq A$.

Now let us consider the cardinality of A and B. Recall that the permuted schedule is constructed through assigning activities by (3.3). Hence all activities in the violating project $\hat{p}$ were assigned the $\hat{p}th$ smallest start time of the corresponding start times in the original schedule. Hence $|A| \leq \hat{p} - 1$. Note that $|A|$ could be smaller than $\hat{p} - 1$ if there exists some $p' \in P_c : p' < \hat{p}$ such that $S^{\pi}_{i,p',c} = S^{\pi}_{i,\hat{p},c}$. Similarly $|B| \geq \hat{p}$.

Finally as $|B| \geq \hat{p} > \hat{p} - 1 \geq |A|$, then $B \nsubseteq A$. Therefore there exists at least one project $p \in P_{\hat{c}}$ such that $p \in B$ and $p \notin A$, hence $S_{i,p,\hat{c}} \geq S^{\pi}_{i,\hat{p},\hat{c}}$ and $S_{i',p,\hat{c}} \leq S^{\pi}_{i',\hat{p},\hat{c}}$, which implies $S^{\pi}_{i',\hat{p},c} - S^{\pi}_{i,\hat{p},c} \geq S_{i',p,c} - S_{i,p,c}$. This is a contradiction to our initial assumption and hence the permuted schedule of a feasible schedule is time-feasible. □

These lemma leads to the following theorem.

Theorem 3.3.3. *A feasible schedule $\mathcal{S}$ is symmetric to its permuted schedule $\mathcal{S}^{\pi}$*

Proof. Equation (3.3) is a symmetry if it maps feasible solutions to other feasible solutions with the same objective value. From Lemma 3.3.1 and Lemma 3.3.2 we know that if $\mathcal{S}$ is feasible then $\mathcal{S}^{\pi}$ is resource-, and time-feasible respectively. Hence $\mathcal{S}^{\pi}$ is also feasible. Finally as $m_0 = 1$ the start time of the dummy end activity, ω, is not permuted and hence $\mathcal{S}$ and $\mathcal{S}^{\pi}$ have the same objective value. Hence equation (3.3) is a symmetry and $\mathcal{S}$ and $\mathcal{S}^{\pi}$ are symmetric. □

A direct-consequence of Theorem 3.3.3 is that we are able to introduce an additional set of precedence constraints between activities of the same index of successive projects of the same class, i.e. $A_1^{sym} := \{((i,p,c),(i,p+1,c))|i \in V_c, p \in P_c \setminus \{m_c\}, c \in C\}$, all with an associated time lag $\delta_{(i,p,c),(i',p+1,c)} = 0$. These additional precedence constraints can be considered to be symmetry-breaking constraints.

Due to the presence of the renewable resources, it may be possible to introduce a second set of additional precedence constraints, which exploit situations where the number of projects in a class exceeds the number of activities of the same index that can be executed in parallel without violating the capacity of the resources. In such cases end-start precedence constraints can be added between certain pairs of activities as follows.

Proposition 3.3.4. *Given the permuted schedule, $\mathcal{S}^\pi$, of some feasible schedule $\mathcal{S}$. If there exists a project class $c \in C$ with activity $i \in V_c$ such that $\epsilon = \min_{k\in\mathcal{R}} \left\lfloor \frac{R_k}{r_{ick}} \right\rfloor \geq 1$, then for all projects $p \in \{0, ..., m_c - \epsilon - 1\}$, $S^\pi_{i,p,c} + d_{i,c} \leq S^\pi_{i,p+\epsilon,c}$.*

Proof. From Theorem 3.3.3, we know that in the permuted schedule, $S_{i,p,c} \leq S_{i,p+1,c}$ for all $i \in V_c, p \in P_c, c \in C$. Now assume for the sake of contradiction for a given activity index $i \in V_c$ from a given class $c \in C$, there exists an integer $z = \min_{k\in\mathcal{R}} \left\lfloor \frac{R_k}{r_{ick}} \right\rfloor \geq 1$ such that $S^\pi_{i,p,c} + d_{i,c} > S^\pi_{i,p+z,c}$ for some $p \in \{1, ..., z\}$. The resource demand $r_k(S^\pi, d_{i,c} - 1) \geq r_{i,c,k} \cdot (z + 1) = (\left\lfloor \frac{R_k}{r_{ick}} \right\rfloor + 1) \cdot r_{ick} > R_k$. Hence the model is not resource-feasible, which is a contradiction. □

Hence, it is possible to add to the model a second set of additional precedence constraints, $A_2^{sym} := \{(i, p, c), (i, p + \epsilon, c) | i \in V_c, p \in \{0, ..., \epsilon - 1\}, c \in C\}$ with associated time lag, $\delta_{(i,p,c),(i,p+\epsilon,c)} = d_{i,c}$. It should be noted that these additional precedence constraints would be propagated by certain cumulative propagation algorithms used in constraint programming if A_1^{sym} were included.

Some multi-project scheduling problems in the literature distinguish between global and local resources. We show here when considering local resources that in general the symmetry no longer exists. The renewable resources we have considered so far, $\mathcal{R}$, can be seen as global resources as they must be shared between all the individual projects. By contrast, local resources are used exclusively by individual projects. With respect to our notation, the local resources $Q_{p,c}$ can be introduced for each project $p \in P_c$ and each class $c \in C$. We assume that all projects of the same class have identical, yet distinct, local resources, hence we say $Q_{p,c} = Q_c$ for all $p \in P_c$. As with global resources, each local resource, $k \in R_c^{local}$, has finite capacity, $R^{local}_{c,k}$, and each activity of the class $i \in V_c$ requires $r^{local}_{i,c,k}$ units. Hence a schedule $\mathcal{S}$ is *local-resource-feasible* if

$$r_k(S, t) = \sum_{i\in V_c : t\in[S_{i,p,c}, S_{i,p,c}+d_{i,c})} r^{local}_{i,c,k} \leq R^{local}_{c,k}, \quad \forall\, t \in H,\ k \in Q_c,\ p \in P_c, c \in C \quad (3.4)$$

Proposition 3.3.5. *The additional symmetry breaking precedence constraints, A_1^{sym}, and consequently A_2^{sym}, are not valid when local resources are considered.*

Proof. The proof follows a simple counter example (see Figure 3.3) for which an optimal solution without the constraints is better than the optimal solution with the constraints. For brevity in this example we ignore the dummy nodes and arcs. Consider scheduling two projects of the same class, i.e. $c = 1 \in C$ and $m_1 = 2$. The class consists of two activities, $n_1 = 2$. Both activities have the same duration $d_{1,1} = d_{1,2} = 1$. No precedences exist between the two activities, $A_1 = \emptyset$. The class considers a single local resource $Q_c = \{1\}$ with capacity, $R_1^{local} = 1$ and one global resource $\mathcal{R} = \{1\}$ with capacity $R_1 = 1$. Finally the activities have the following resource requirements, $r_{1,1,1} = 1$, $r_{2,1,1} = 0$ and $r_{1,1,1}^{local} = r_{2,1,1}^{local} = 0$.

Consider an optimal schedule, $\mathcal{S}$, where $S_{1,1,1} = S_{2,2,1} = 0$ and $S_{2,1,1} = S_{1,2,1} = 1$. This schedule does not violate the local resources as activities of the same project are never executed concurrently. The schedule does not violate the global constraint as the activities of index 1 are never executed concurrently. Now consider the permuted version of this schedule. The global resource is still feasible, the two activities of index 1 have not changed. However the capacity of both local resources are exceeded. Furthermore when the precedence constraints are considered between the activities of the same index of projects from the same class the optimal solution of three can be found where $S_{1,1,1} = 0$, $S_{2,1,1} = S_{1,2,1} = 1$ and $S_{2,2,1} = 2$, which is clearly worse than when these precedence constraints were not considered. □

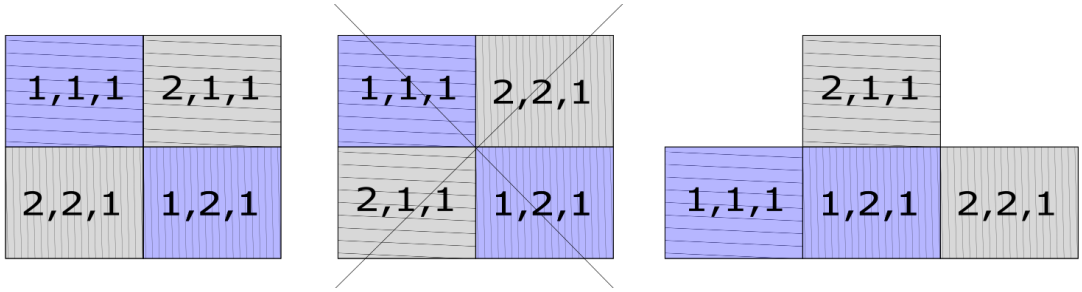


Figure 3.3: A visualisation of the counter example used in the proof of Proposition 3.3.5. It shows the symmetry breaking constraints are not valid for projects with local resources. Activities from the same project have the same pattern. Both activities require the local resource so cannot overlap. The two activities using the global resource are coloured blue and cannot overlap. The left image represents the optimal solution without the symmetry breaking constraints, which takes 2 time-units. The middle image represents the permuted schedule, which is infeasible since it violates both local resources. The right image represents the optimal schedule when the additional precedences constraints are added, which takes 3 time-units.

3.4 Mixed-Integer Programming

In this section we first show how a standard MIP model for RCPSP based on binary pulse start decision variables can be reformulated for the high-multiplicity RCPSP/max. We then show how a version of this model can be formulated that takes advantage of the symmetry between projects from the same class by using integer variables instead of binary. By considering integer variables, we reduce the total number of variables being considered by the model. We then propose another reduced model that is based on different decision variables. A number of tightening constraints are proposed to ensure that the continuous relaxations are at least as good as a simple resource-based lower bound.

To narrow the scope of analysis we consider MIP models with discrete-time start variables and precedence constraints that are in disaggregated form. Discrete-time models with disaggregated precedence constraints are known to have the strongest LP relaxations compared to alternative MIP models, as summarised by Artigues (2017). Furthermore they are known to perform well in practice, as verified by both Tesch (2018a) and Bianco and Caramia (2013). This is not to say that discrete-time models are necessarily the best models for all instances. Kopanos et al. (2014) compare a number of different MIP approaches to the RCPSP and find empirically two of their formulations based on continuous variables perform on average the best across a range of instances. Koné et al. (2011) claim that event-based and flow-based formulations outperform the discrete time-based methods when the time horizon is very large.

All models require an upper bound on the duration of the project, T. The time-horizon of the project is then $H := [0, ..., T[$. The earliest start time, $ES_{i,p,c}$, of activity $(i, p, c) \in V$, is equal to the longest path from the dummy start node to (i, p, c) in the project network. Similarly the latest start time, $LS_{i,p,c}$ is equal to $T - 1$ minus the longest path from from (i, p, c) to the dummy end node.

3.4.1 Formulation based on binary pulse start variables

With respect to the classical RCPSP, the discrete-time model was introduced in the pioneering work of Pritsker et al. (1969). The discrete time model was extended to the discrete time model with disaggregated precedences by Christofides et al. (1987), which provide stronger continuous relaxations. With respect to our notation, the model is based on binary variable $x_{i,p,c,t}$, for all $(i, p, c) \in V$ and $t \in H$ such that $x_{i,p,c,t} = 1$ if and only if activity (i, p, c) starts at time t and 0 otherwise. We

generalise this model to take into account the generalised precedence constraints. This type of decision variable is known as *pulse variables* and the basic discrete-time formulation based on pulse variables, denoted DDT, is then modelled by:

$$\min \quad \sum_{t \in H} t \cdot x_{\omega,t} \tag{3.5a}$$

$$\text{s.t.} \quad \sum_{\tau=0}^{t-\delta_{(i,p,c),(\bar{i},\bar{p},\bar{c})}} x_{i,p,c,\tau} - \sum_{\tau=0}^{t} x_{\bar{i},\bar{p},\bar{c},\tau} \geq 0 \qquad \forall\, ((i,p,c),(\bar{i},\bar{p},\bar{c})) \in A;\ t \in H \tag{3.5b}$$

$$\sum_{(i,p,c) \in V} \sum_{\tau=t-d_{i,c}+1}^{t} r_{i,c,k} \cdot x_{i,p,c,\tau} \leq R_k \qquad \forall\, t \in H;\ k \in \mathcal{R} \tag{3.5c}$$

$$\sum_{t \in H} x_{i,c,p,t} = 1 \qquad \forall\, (i,p,c) \in \mathcal{V} \tag{3.5d}$$

$$x_{i,p,c,t} = 0 \qquad \forall\, (i,p,c) \in \mathcal{V};\ t \in H \setminus [ES_{ipc}, LS_{ipc}] \tag{3.5e}$$

$$x_{i,p,c,t} \in \{0,1\} \qquad \forall\, (i,p,c) \in \mathcal{V};\ t \in [ES_{ipc}, LS_{ipc}] \tag{3.5f}$$

The disaggregated generalised precedence constraints (3.5b) enforce the logical relation: $S_{i',p',c'} \leq t \implies S_{i,p,c} \leq t - \delta_{(i,p,c),(i',p',c')}$. Constraints (3.5c) enforce the resource constraints, (3.5d) ensure that each activity starts in the time horizon, (3.5e) ensure that the activities do not start before their earliest starting time but not later than their latest starting time. Finally constraints (3.5f) define the binary pulse decision variables.

The symmetry contained in DDT can be removed by considering the additional sets of precedence constraints $A \cup A_1^{sym} \cup A_2^{sym}$ in constraints (3.5b). We will refer to the DDT model with additional symmetry breaking constraints as DDT-Sym. Alternatively the symmetry in the model can be broken by remodelling the problem, as discussed next.

3.4.2 Reduced formulation based on integer pulse start variables

We now propose a reduced version of the pulse start model based on integer variables $y_{i,c,t}$ for all $i \in V_c$ and $t \in H$ for projects of class $c \in C$, such that $y_{i,c,t}$ is an integer variables that indicates the number of activities $i \in V_c$ from any project $p \in P_c$ for class $c \in C$ that start at time t.

This model takes advantage of the symmetry between projects from the same

class by no longer keeping track of exactly which specific project of the same class an activity comes from. For any feasible solution a specific assignment of activities to projects and be created by assigning activities with earlier start times to projects with lower indices. The decision variables are integer, as opposed to binary, as it may be possible that two or more activities of the same index from projects of the same class start at the same time in a feasible schedule. To restrict the upper bound of these integer variables, we introduce the parameter $\mu_{i,c} := \min(m_c, \min_{k\in\mathcal{R}} \lfloor \frac{R_k}{r_{i,c,k}} \rfloor)$, which represents the maximum number of multiples of activity $i \in V_c$ available from class $c \in C$ that can occur simultaneously without violating any resource constraints. Furthermore as precedences are now only defined between activities from the same project, and not also used for symmetry breaking constraints, we defined the short-hand time lag notation $\delta_{i,i',c} := \delta_{(i,p,c),(i',p',c')}$ for all $(i,i') \in A_c$ for project class $c \in C$.

Hence the reduced model based on integer pulse variables, denoted RDDT, is:

$$\min \quad \sum_{t\in H} t \cdot y_{\omega,t} \tag{3.6a}$$

$$\text{s.t.} \quad \sum_{\tau=0}^{t-\delta_{i,i',c}} y_{i,c,\tau} - \sum_{\tau=0}^{t} y_{i',c,\tau} \geq 0 \qquad \forall\, (i,i',c) \in A_c;\ c \in C;\ t \in H \tag{3.6b}$$

$$\sum_{c\in C} \sum_{i\in V_c} \sum_{\tau=t-d_{i,c}+1}^{t} r_{i,c,k} \cdot y_{i,c,\tau} \leq R_k \qquad \forall\, t \in H;\ k \in \mathcal{R} \tag{3.6c}$$

$$\sum_{t\in H} y_{i,c,t} = m_c \qquad \forall\, i \in V_c;\ c \in C \tag{3.6d}$$

$$\sum_{\tau=0}^{t} y_{n_c+1,c,\tau} - m_c \sum_{\tau=0}^{t} y_{\omega,\tau} \geq 0 \qquad \forall\, c \in C;\ t \in H \tag{3.6e}$$

$$y_{i,c,t-d_{i,c}} \leq \mu_{i,c} \sum_{\tau=t}^{T} y_{\omega,\tau} \qquad \forall\, i \in X_c;\ c \in C;\ t \in H \tag{3.6f}$$

$$y_{i,c,t} \in [0, ..., \min(\mu_{i,c}, T_{i,c,t})] \qquad \forall\, i \in V_c;\ c \in C;\ t \in H \tag{3.6g}$$

The objective (3.6a) is to minimise the starting time of the dummy end activity. Constraints (3.6b) represent the disaggregated generalised precedence constraints and work by ensuring that the number of activities i that have started by $t - \delta_{i,i',c}$ must be at least equal to the number of activities i' that have started by time t. Constraints (3.6c) are the resource constraints. Constraints (3.6d) state that the number of times the activities occurs is equal to the multiplicity of the class that the activity belongs to. Constraints (3.6e) are disaggregated precedence constraints between

the project dummy end nodes and the master dummy end nodes. Constraints (3.6f) are aggregated precedence constraints for all activities that may finish last in a class, $X_c := \{i \in V_c \mid (i,\omega) \in A_c \wedge \mu_{i,c} < m_c\}$. Finally (3.6g) are the integer pulse start variables, where $T_{i,c,t} = |\{p \in P_c \mid t \in [ES_{i,p,c}, LS_{i,p,c}]\}|$

If all the individual projects are modelled as single instances of a distinct project class, then $RDDT$ and DDT are equivalent. This can be seen as constraints (3.6e) become disaggregated precedence constraints with a time lag of zero and no constraints of type (3.6f) are added when a single multiple is considered. Furthermore the definition of (3.6g) enforces that activities start between their earliest and latest start times. Given this relationship we say DDT is the expanded version of $RDDT$ and $RDDT$ is the reduced version of DDT.

3.4.3 Reduced formulation based on integer step & on-off variables

In this section we introduce an alternative MIP model that uses both step variables and on/off style start variables as opposed to pulse start variables. Here we present only the reduced version of the model, which utilises integer variables and is denoted $ROOSDDT$. Similar to the relationship between $RDDT$ and DDT, it is possible to obtain an equivalent expanded version of the model if all the individual projects are modelled as single instances of a distinct project class. The expanded version of $ROOSDDT$ is denoted $OOSDDT$. Furthermore it is possible to break the symmetry in $OOSDDT$ by considering additional symmetry breaking constraints A_1^{sym} and consequently A_2^{sym}. We denote this model OOSDDT-Sym.

Step variables were first introduced as an alternative to pulse start variables for the RCPSP (Pritsker et al., 1969). Compared with pulse-start variables, step-start variables greatly reduce the density, i.e., the number of non-zero terms in the constraint matrix, of the resulting MIP model. As discussed by Sankaran et al. (1999), step variables had the additional benefit for the classical RCPSP in that when the resource constraints are relaxed the constraint matrix is totally unimodular.

We define our step variables as follows. Let $z_{i,c,t}$ equal the number of activities of index $i \in V_c$ from projects $p \in P_c$ of class $c \in C$ that have started by time $t \in H$. Hence $z_{i,c,t} = \sum_{\tau=0}^{t} y_{i,c,\tau}$.

A practical downside of using step-variables in isolation is that in our experience commercial MIP packages are currently not able to automatically generate as many cuts during solve as they can with pulse starts and on-off starts. To overcome

this we augment our step-variable models with addition on/off variables. On/off variables were first motivated by the RCPSP where preemption was allowed (Kaplan, 1998). However on/off variables have recently been shown to perform well in practice in conjunction with other variable types for the RCPSP by Kopanos et al. (2014). To reduce the number of these additional variables that are required, we first define two additional sets of activities.

First let Ψ be the set of all the different resource requirements of the nodes, i.e., $\Psi = \bigcup_{i \in V_c} \bigcup_{c \in C} (\{r_{i,c,k} | k \in \mathcal{R}\})$. Each set of resource requirements $\psi \in \Psi$ defines the amount of resources that at least one activity requires, $r_{\psi,k}$ for all $k \in R$. Hence only one on-off variable is introduced for all activities with the same resource requirements. Furthermore for each $\psi \in \Psi$ we define a set of activities $\overline{V}_\psi := \{(i,p,c) \in V | \bigwedge_{k \in R} r_{i,c,k} \equiv r_{\psi,k}\}$. It can be seen that in the worst case, each activity has a unique resource requirement across the set of renewable resources and hence $|\Psi| = |V|$.

We define our on-off variables as follows. Let $\xi_{\psi,t}$ equal the number of activities with resource requirement $\psi \in \Psi$ that are being processed at time $t \in H$. Hence $\xi_{\psi,t} = \sum_{(i,p,c) \in \overline{V}_\psi} (z_{i,c,t} - z_{i,c,t-d_{i,c}})$ for all $\psi \in \Psi$. Now the reduced model based on integer step and on/off variables, denoted by ROOSDDT, is given by,

$$\min \quad \sum_{t \in H} t \, (z_{\omega,t} - z_{\omega,t-1}) \tag{3.7a}$$

$$\text{s.t.} \quad z_{i,c,t-\delta_{i,i',c}} - z_{i',c,t} \geq 0 \qquad \forall \, (i,i') \in A_c; \; c \in C; t \in H \tag{3.7b}$$

$$\xi_{\psi,t} = \sum_{(i,p,c) \in \overline{V}_\psi} (z_{i,c,t} - z_{i,c,t-d_{i,c}}) \qquad \forall \, \psi \in \Psi; \; t \in H \tag{3.7c}$$

$$\sum_{\psi \in \Psi} r_{\psi,k} \cdot \xi_{\psi,t} \leq R_k \qquad \forall \, t \in H; \; k \in \mathcal{R} \tag{3.7d}$$

$$z_{n_c+1,c,t} - m_c \cdot z_{\omega,t} \geq 0 \qquad \forall \, c \in C; \; t \in H \tag{3.7e}$$

$$z_{i,c,t+1} - z_{i,c,t} \leq \mu_{i,c} \cdot (1 - z_{\omega,t+d_{i,c}}) \qquad \forall \, i \in X_c, c \in C, t \in H \tag{3.7f}$$

$$z_{i,c,t} \geq z_{i,c,t-1} \qquad \forall \, i \in V_c; \; c \in C; \; t \in H \tag{3.7g}$$

$$z_{i,c,t} \in [T^{min}_{i,c,t}, ..., T^{max}_{i,c,t}] \qquad \forall i \in V_c; \; c \in C; \; t \in H \tag{3.7h}$$

$$\xi_{\psi,t} \in [0, ..., \mu_\psi] \qquad \forall \psi \in \Psi; \; t \in H \tag{3.7i}$$

Note that in the way that we have presented constraints (3.7a-3.7c, 3.7f-3.7g) it is possible that t can be defined outside of the time-horizon. We assume $z_{i,c,t} = 0$ if $t \notin H$ where required. The objective (3.7a) is to minimise the starting time of the dummy end activity. Constraints (3.7b) represent the disaggregated generalised

precedence constraints. Constraints (3.7c) are the consistency constraints between the two variables types. Constraints (3.7d) are the resource constraints based on the on/off variables. Constraints (3.7e) ensure the dummy end starts after all the projects are completed similar to (3.6e). Constraints (3.7f) are aggregated precedence constraints similar to (3.6f). Constraints (3.7g) are consistency constraints for the step variables. Constraints (3.7h) define the integer step variables between the bounds $T^{min}_{i,c,t} := |\{p \in P_c | LS_{i,p,c} \leq t\}|$ and $T^{max}_{i,c,t} := |\{p \in P_c | ES_{i,p,c} \geq t\}|$. The definition of $T^{min}_{i,c,t}$ ensures that all the activities are completed by the end of the time horizon. Finally constraints (3.7i) define the on-off integer variables.

3.4.4 Synthesis of polyhedral analysis

An important method of comparing different MIP models is based on the strength of their respective linear programming relaxations. Artigues (2017) clarifies the current knowledge about the strengths of the different formulations to time-index RCPSP models. While acknowledging that practical performances of integer programming is not necessarily related to the strength of the LP relaxation, Artigues (2017) argues that “new" formulations that are equivalent to existing formulations via non-singular transformations should be distinguished from formulations with stronger linear relaxations. Therefore in this section we compare the different MIP formulations presented in this chapter with respect to the DDT, which is well known in the literature.

We recall the following definitions as given by Artigues (2017) for any pair of integer linear programming formulations F_1 and F_2 and their respective linear relaxations $P(F_1)$ and $P(F_2)$:

- $F_1 \equiv^{LP} F_2$ if there exists an affine non-singular (i.e. bijective) transformation that allows one to obtain one formulation from the other.
- $F_1 \preceq^{LP} F_2$ if there exists an affine transformation of F_2 on the solution space $P(F_1)$ that gives formulation F_2' and solution space $P(F_2')$ such that $P(F_2') \subseteq P(F_1)$.
- $F_1 \prec^{LP} F_2$ if $F_1 \preceq^{LP} F_2$ and if, in addition, we can find a point $x \in P(F_1)$ such that $x \notin P(F_2')$ which yields $P(F_2') \subset P(F_1)$.

Hence, if $F_1 \equiv^{LP} F_2$, then F_1 and F_2 have the same relaxation strength and thus provide the same LP relaxation (lower) bound of the integer program. Furthermore if $F_1 \prec^{LP} F_2$, then there exists an objective coefficient vector such that the LP lower bound (for minimisation) of F_1 is strictly lower than the lower bound of F_2.

The following proposition summarises the relative strengths of the linear relaxation of the various models considered in this chapter. Here we simply state the proposition. For brevity the proofs are omitted from the chapter but can be found in Appendix A.2.

Proposition 3.4.1. *For the total-makespan objective*

$$\begin{aligned} RDDT \equiv^{LP} ROOSDDT \prec^{LP} \\ DDT \equiv^{LP} OOSDDT \prec^{LP} \\ DDT\text{-}Sym \equiv^{LP} OOSDDT\text{-}Sym \end{aligned}$$

From Proposition 3.4.1 we know that in general the linear-programming relaxation of the models based on integer variables are not as strong as that of the binary variables. Furthermore the symmetry breaking constraints strengthen the linear relaxations. Finally the models based on the step and on-off variables are equivalent up to non-singular transformation of the corresponding version of the model.

3.5 Constraint Programming

Constraint programming offers an alternative methods of modelling optimization problems. For completeness we include the CP models used by the CP solvers in the computational study. These models are already known in the literature, however the models allow us to explain explicitly how the symmetry breaking constraints are added.

3.5.1 Integer Based Variables

The starting time of each activity $(i, p, c) \in V$ is assigned a decision variable $S_{i,p,c}$, which must be mapped to an integer from the initial domain $X^{init}_{i,p,c} := [ES_{i,p,c}, LS_{i,p,c}]$. A typical CP model for the RCPSP/max is then,

$$\begin{aligned} \min \quad & S_\omega & & \\ \text{s.t.} \quad & S_{i,p,c} + \delta_{(i,p,c),(i',p',c')} \leq S_{i',p',c'} & \forall\, ((i,p,c),(i',p',c')) \in A & \quad (3.8a) \\ & \texttt{cumulative}(s,\ d,\ [r_{i,c,k}(i,p,c) \in V],\ R_k) & \forall\, k \in R & \quad (3.8b) \\ & S_{i,p,c} \in \texttt{integer}(X^{init}_{i,p,c}) & \forall\, (i,p,c) \in V & \quad (3.8c) \end{aligned}$$

Constraints (3.8a) are simple difference constraints that enforce the generalised precedence constraints. Each resource is represented by a cumulative constraint (3.8b) with a constant capacity over the considered project duration. Constraints (3.8c) ensure that the decision variables take a value from their domain. As discussed in Chapter 2 of this thesis, there are many propagation algorithms for the cumulative constraint, but the most widely used for project scheduling problems is based on timetable propagation (Pape, 1994), which is used here.

Additional constraints can be added to the model to improve the strength of propagation. We say that two activities, (i, c) and (i', c'), are in *disjunction* if they cannot be executed in parallel, i.e., $r_{i,c,k} + r_{i',c',k} > R_k$ for some $k \in R$. Let D be the set of all pairs of disjunctive activities. In some circumstances, it is possible to infer the ordering of a pair of disjunctive activities. More explicitly, we define the subset of disjunctive activities for which an ordering can be inferred as $\overline{D} := \{((i,p,c),(i',p',c)) \in D |\ LS_{i,p,c} < ES_{i',p',c'} + d_{i',c'}\}$. Here we are effectively implementing the disjunctive constraint, discussed in Chapter 2, by the two half-reified constraints (Feydy et al., 2011) sharing some boolean variable $B_{(i,p,c),(i',p',c')}$. The additional constraints are as follows:

$$S_{i,p,c} + d_{i,c} \leq S_{i',p',c'} \qquad \forall\ ((i,p,c),(i',p',c')) \in \overline{D} \tag{3.9a}$$

$$B_{(i,p,c),(i',p',c')} \rightarrow S_{i,p,c} + d_{i,c} \leq S_{i',p',c'} \qquad \forall\ ((i,p,c),(i',p',c')) \in D \setminus \overline{D} \tag{3.9b}$$

$$\neg B_{(i,p,c),(i',p',c')} \rightarrow S_{i',p',c'} + d_{i',c'} \leq S_{i,p,c} \qquad \forall\ ((i,p,c),(i',p',c')) \in D \setminus \overline{D} \tag{3.9c}$$

$$B_{(i,p,c),(i',p',c')} \in \texttt{boolean} \qquad \forall\ ((i,p,c),(i',p',c')) \in D \setminus \overline{D} \tag{3.9d}$$

Constraints (3.9a) add precedence constraints between the ordered, disjunctive activities. Constraints (3.9b) and (3.9c) enforce that for each pair of disjunctive activities, one more end before the other starts. Finally (3.9d) define the additional boolean variables. It is possible to remove the symmetry from this model by considering the additional arc set $A \cup A_1^{sym}$ in constraints (3.8a).

3.5.2 Interval Based Variables

For completeness, we include the scheduling-specific model possible in CP Optimizer, which makes use of interval variables. The model is based on a single type of interval variable, $\alpha_{i,p,c}$, representing the activity of index $i \in V_c$ of project $p \in P_c$ of class $c \in C$. The duration of the interval variables are fixed to the duration of the activities. Each renewable resource $k \in R$ is represented by a single cumulative function, f_k.

The model, denoted by CpOpt, is given by,

$$\min \quad \max_{(i,p,c)\in V} \left(\texttt{endOf}(\alpha_{i,p,c})\right) \tag{3.10a}$$

$$\text{s.t.} \quad \texttt{startBeforeStart}(\alpha_{i,p,c}, \alpha_{i',p',c'}, \delta_{(i,p,c),(i',p',c')}) \quad \forall\, ((i,p,c),(i',p',c')) \in A \tag{3.10b}$$

$$f_k = \sum_{(i,p,c)\in V} \texttt{pulse}(\alpha_{i,p,c}, r_{i,c,k}) \leq R_k \quad \forall\, k \in R \tag{3.10c}$$

$$\alpha_{i,p,c} \in \texttt{interval}(comp, d_{i,c}) \quad \forall\, (i,p,c) \in V \tag{3.10d}$$

$$f_k \in \texttt{cumul} \quad \forall k \in R \tag{3.10e}$$

The objective (3.10a) is to minimise the maximum end time of each interval variable. Constraints (3.10b) represent the generalised precedence constraints. Constraints (3.10c) represent the resource constraints by implicitly building a cumulative expression. The pulse function, $\texttt{pulse}(\alpha, r)$ enforces the cumulative resource expression by stating the function is increased by r units at the start of interval α and decreased by r at the end of interval α. The interval variables and cumulative functions are defined in (3.10d) and (3.10e), respectively. The symmetry that exists in CpOpt can be removed by considering the additional set of precedence constraints $A \cup A_1^{sym}$ in constraints (3.10b). We will refer to the CpOpt model with additional symmetry breaking constraints as CpOpt-Sym.

3.6 Computational study

The computational study consists of two parts. Firstly, we seek to understand the effects of the different approaches of symmetry breaking on the presented MIP models as well as some state-of-the-art CP-based methods on the CP models. As the high-multiplicity RCPSP/max has not been considered in the literature, there does not exist any publicly available datasets. To construct a data set, we take multiples of instances from the well-known Project Scheduling Problem Library (PSPLIB) (Kolisch and Sprecher, 1997). Secondly, to compare the best performing approaches, we consider relevant instances from the Multi-Project Scheduling Library (MPSPlib) (Homberger, 2007). These instances do not consider generalised precedence constraints and are thus a special case of the high-multiplicity RCPSP/max.

Both experiments were run on the MonARCH HPC Cluster. The processors are Intel Xeon E5-2667 v3 3.2GHz, 20M Cache, 9.60GT/s QPI, Turbo, HT, 8C/16T (135W). The memory and number of CPUs requested differ for the two experiments

and will be discussed in the relevant sections. Gurobi 7.5 was used for all MIP models. Dual simplex and barrier methods were run concurrently to solve the root node of the MIP models. Two different commercial CP-solvers were used: Opturion's Discrete Optimiser 1.0.2 (Cpx) and IBM CPLEX 12.8.0 CP Optimizer. The other CP solver used is *Chuffed*. Both Chuffed and Cpx solvers were executed on the Integer Based CP model, outlined in Section 3.5.1, implemented on Minizinc-2.1.7 (Nethercote et al., 2007). CP Optimizer used the Interval Based CP Model, outlined in Section 3.5.2.

3.6.1 Multiples of PSPLIB

Solution methods

Table 3.1 summarises the different solution methods being tested in this section. In total there are twelve different approaches being tested that can be divided into five distinct families. The first two families are distinguished based on the type of decision variables considered in the corresponding MIP models, pulse start variables and step & on-off variables respectively. For each of two families, the expanded version is tested without symmetry breaking constraints, the expanded version is tested with symmetry breaking constraints, and finally the reduced version of the model is tested. The MIP models include addition tightening constraints that are detailed in Appendex A.1.

Table 3.1: A summary of the different methods being tested. Symmetry conditions (Sym): (0) there exists symmetry in the model, (1) symmetry is removed through additional precedence constraints, (2) symmetry is removed through remodelling

Family	Acronym	Sym	Section	Formulation	Solver
Pulse MIPs	DDT	0	3.4.1	MIP (Pulse)	Gurobi
	DDT-Sym	1	3.4.1	MIP (Pulse)	Gurobi
	RDDT	2	3.4.2	MIP (Reduced Pulse)	Gurobi
Step MIPs	OOSDDT	0	3.4.3	MIP (Step)	Gurobi
	OOSDDT-Sym	1	3.4.3	MIP (Step)	Gurobi
	ROOSDDT	2	3.4.3	MIP (Reduced Step)	Gurobi
Chuffed	Chuffed	0	3.5.1	CP (Integer)	Chuffed
	Chuffed-Sym	1	3.5.1	CP (Integer)	Chuffed
Cpx	Cpx	0	3.5.1	CP (Integer)	Opturion Cpx
	Cpx-Sym	1	3.5.1	CP (Integer)	Opturion Cpx
CpOpt	CpOpt	0	3.5.2	CP (Interval)	CP Optimizer
	CpOpt-Sym	1	3.5.2	CP (Interval)	CP Optimizer

The remaining three families of approaches are distinguished based on the CP solver used; Chuffed, Cpx, and CP Optimizer. For each there are two approaches: 0 or 1 (-Sym suffix) for solving without or respectively with the symmetry breaking constraints. The authors of Schutt et al. (2013c) mentioned that it was impossible to test symmetry breaking methods in their approach but recommended Chuffed.

To stay as consistent with the literature as possible for Chuffed we adopted the search strategy that obtained the best results in Kreter et al. (2017). It alternates between two different search strategies. The first is a *first fail* approach, which selects the variable with the smallest domain size and assigns the minimal value in the domain to it. The second search strategy is an activity-based search, which is a variant of the Variable-State-Independent-Decaying-Sum (VSIDS) approach developed for SAT solvers (Moskewicz et al., 2001). The search is combined with Luby restarts (Luby et al., 1993) and a restart base of 100 conflicts. Hence the search strategy alternates between the two approaches after each restart using the same restart policy and base as for activity-based search.

Opturion's Cpx is similar to Chuffed in the sense that it uses lazy clause generation techniques to learn from failure during search as well as using Luby restarts. For optimization problems, the default search is preceded by a probing phase in which random probes of the search tree are used to initialise the variable activity counts.

Experimental setup

Given the lack of publicly available data sets of the high-multiplicity RCPSP/max, we artificially construct instances with symmetry by taking multiples of the 73 feasible instances of the UBO10 data set from PSPLIB. Each instance in this data set considers 10 activities and 5 resources. We treat each instance of the UBO10 problem as a project class. All 12 of the scheduling methods are compared on instances containing a single project class and multiplicities taken from the range $\{2, 3, 5, 8, 10\}$. Hence each approach is tested on a total of 365 instances. Admittedly, it would be realistic to have multiple classes per instance. Such situations are considered in the subsequent computational study. To align with the specifications used in recent papers that consider the RCPSP/max instances in PSPLIB (Schutt et al., 2013c; Vilim et al., 2015) each approach is given 10 minutes wall time on a single CPU. Finally 4GB RAM is requested.

As each of the 73 feasible instances of the UBO10 test set have already been proven optimal, these optimal values are used to construct valid upper bounds.

A feasible canonical schedule is obtained by sequentially scheduling the correct number of multiples of these optimal solutions one after the other. The makespan of the canonical schedule is used as the upper bound for all the models. Furthermore the MIP models are seeded with the canonical schedule to ensure they at least obtain a feasible solution.

Results / discussion

Table 3.2 summarises the number of instances across the varying multiplicities for which the different methods in the different symmetry conditions could (1) prove optimality and (2) find solutions that are known to be optimal.[1] The symmetry breaking techniques allow each of the approaches to both solve more instances to optimality and find more instances that are known to be optimal.

The results suggest it is more effective to break the symmetry by reformulating the models to their reduced versions rather than adding additional precedence constraints to the expanded versions of the models. This is despite the stronger formulation of the expanded model and the expanded model with additional symmetry breaking constraints. Remodelling allows fewer variables to be considered, albeit integer instead of binary, rather than simply adding more constraints to represent the additional precedences. However the integer variable models are still restricted by the increasing time-horizon resulting from the increased project multiplicities and hence for multiplicities of 5, 8 and 10 find and prove very few optimal solutions. This is further highlighted by noticing that for three of the instances the canonical solution is in fact the optimal solution.

The CP-based methods all benefit greatly from the symmetry breaking constraints. Cpx-Sym appears to be the best in proving optimality, managing to prove 210 instances optimal and 10 instances with a multiplicity of 10. Chuffed-Sym manages to prove 188 instances optimal, whereas CpOpt-Sym proves 173 optimal. On the other hand, CpOpt manages to find more optimal solutions (218), compared with 204 by Chuffed and 217 by Cpx.

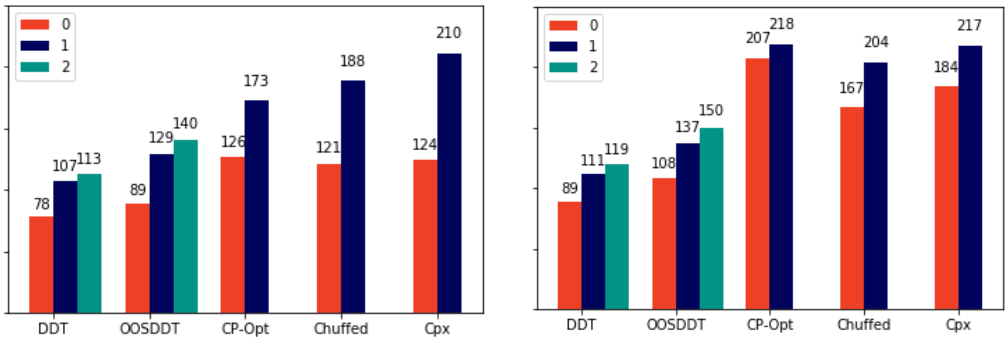
Figures 3.4a and 3.4b summarise the total number of instances for which the methods prove optimality and find optimal solutions respectively. Both figures highlight the benefits of symmetry breaking in increasing the number of instances proven optimal and for finding the known optimal solutions. In particular, for the MIP models it was consistently seen that the reduced models were more effective

[1]The full data and results can be found as the following link: https://doi.org/10.26180/5bfb37e6250ab

Table 3.2: Summary of the number of instances proven optimal / found.

Model	Nb. Optimal Solutions Proven / Found					
	2	3	5	8	10	Total
DDT	68 / 70	10 / 19	0 / 0	0 / 0	0 / 0	78 / 89
DDT-Sym	**73 / 73**	32 / 36	2 / 2	0 / 0	0 / 0	107 / 111
RDDT	72 / 72	**33 / 38**	**6 / 7**	**1 / 1**	**0 / 1**	**113 / 119**
OOSDDT	71 / 62	12 / 34	0 / 1	0 / 1	0 / 0	89 / 108
OOSDDT-Sym	72 / 73	53 / 59	4 / 5	0 / 0	0 / 0	129 / 137
ROOSDDT	**73 / 73**	**57 / 64**	**9 / 12**	**1 / 1**	0 / 0	**140 / 150**
Chuffed	**73 / 73**	46 / 70	2 / 18	0 / 4	0 / 2	121 / 167
Chuffed-Sym	**73 / 73**	**67 / 72**	**33 / 39**	**10 / 12**	**5 / 7**	**188 / 204**
Cpx	**73 / 73**	48 / 72	3 / 29	0 / 7	0 / 3	124 / 184
Cpx-Sym	**73 / 73**	**71 / 72**	**42 / 46**	**14 / 15**	**10 / 11**	**210 / 217**
CpOpt	72 / 73	52 / 72	2 / 41	0 / 14	0 / 7	126 / 207
CpOpt-Sym	**73 / 73**	**67 / 72**	**28 / 46**	**3 / 16**	**2 / 11**	**173 / 218**

than adding additional symmetry breaking constraints to the expanded models.



(a) Solutions proven to optimality

(b) Optimal solutions found

Figure 3.4: A summary of the total number of instances out of the 365 tested for which the solvers managed to (a) prove optimality and (b) find the known optimal solution. Symmetry conditions: (0) no symmetry breaking, (1) symmetry breaking through additional precedences, (2) symmetry breaking through reformulation.

Understanding how each of the methods manages to improve both the upper and lower bounds provides useful insight. Firstly, let us define the notion of *lower bound improvement* and *upper bound improvement*. For a given instance let LB_R be the resource-based lower bound (described in A.1), UB_C be the upper bound obtained by the canonical solution, LB_X and UB_X be the lower bound and upper bound obtained by the solving approach under consideration for that instance respectively, and finally LB_B and UB_B be the best known lower bound and upper bound for that

Table 3.3: The mean lower bound and upper bound improvements for project instances at different multiplicities

Model	Lower Bound Improvement (%)					Upper Bound Improvement (%)				
	2	3	5	8	10	2	3	5	8	10
DDT	98.6	64.7	1.1	0.0	0.0	98.7	48.4	8.1	4.1	4.1
DDT-Sym	100.0	83.7	8.6	0.7	0.6	**100.0**	71.9	8.2	4.1	4.1
RDDT	**99.8**	**88.3**	**64.0**	**24.8**	**11.4**	**99.8**	**81.6**	**35.0**	**6.9**	**5.5**
OOSDDT	99.9	70.5	5.1	0.0	0.0	99.6	67.2	16.1	4.8	4.1
OOSDDT-Sym	**100.0**	91.3	25.8	0.1	0.0	**100.0**	89.7	22.6	4.9	4.1
ROOSDDT	**100.0**	**95.7**	**71.5**	**32.0**	**12.5**	**100.0**	**95.8**	**41.2**	**9.1**	**5.2**
Chuffed	**100.0**	63.0	2.7	0.0	0.0	**100.0**	99.1	84.1	49.3	32.4
Chuffed-Sym	**100.0**	**91.8**	**45.2**	**13.7**	**6.8**	**100.0**	**99.5**	**88.4**	**62.7**	**52.9**
Cpx	**100.0**	65.8	4.1	0.0	0.0	**100.0**	**99.5**	84.1	49.3	32.4
Cpx-Sym	**100.0**	**97.3**	**57.5**	**19.2**	**13.7**	**100.0**	**99.5**	**91.1**	**67.8**	**58.9**
CpOpt	98.7	71.4	5.0	0.8	0.7	**100.0**	**99.5**	**91.4**	**71.4**	**64.7**
CpOpt-Sym	**100.0**	**93.6**	**45.1**	**11.5**	**9.3**	**100.0**	**99.5**	91.3	**71.4**	64.5

instance respectively. Hence the lower bound improvement of a given method on a specific instance is expressed as $LB_{improve} := \frac{\max(LB_X, LB_R) - LB_R}{UB_B - LB_R} * 100$. Likewise the upper bound improvement of a given instance is expressed as $UB_{improve} = \frac{UB_C - UB_X}{UB_C - LB_B} * 100$. If for some instance $UB_C = LB_B$ then we set $UB_{improve} = 100$.

Chuffed and Cpx prove optimality by showing that there does not exist any feasible solution when the time horizon is set to $UB_B - 1$. Hence for both of these solvers we set $LB_X = UB_B$ if $UB_X = LB_X$ otherwise $LB_X = LB_R$. Alternatively it is possible to implement a destructive lower bound, however this is outside the scope of this thesis. Table 3.3 summarises both the average lower bound improvement and upper bound improvement for all instances by multiplicity for each solving method and relevant symmetry condition.

For both DDT and OOSDDT, the reduced formulations significantly improve the lower bound. In fact the ROOSDDT is an efficient method at improving the lower bounds for larger problems. For a multiplicity of 8, the corresponding lower bound improvement of ROOSDDT of 32.0% is significantly better than the other methods. However it is apparent for increasing multiplicities the MIP models become highly inefficient at improving the upper bound. This is true for even the reduced models.

The symmetry breaking constraints considerably help all the CP-based ap-

proaches improve their lower bounds, particularly for problems with fewer projects. Intuitively this makes sense as the additional constraints reduce the size of the search tree. For Cpx and Chuffed this is a direct consequence of being able to solve more models to optimality. For CpOpt, the symmetry breaking constraints seem to assist the failure directed search in improving the lower bound. It is likely that the lower bounds obtained by Chuffed and Cpx could be significantly improved if, like CpOpt, they switched to trying to improve the lower bound once a good quality feasible solution is found.

Interestingly, in general the symmetry breaking constraints did not seem to help CpOpt find better feasible solutions. For multiplicities 5 and 10, CpOpt on average found marginally better feasible solutions without the symmetry breaking constraints than with them. This appears to contradict the results found for the more practical problems reported in Edwards et al. (2017) and Kovács and Váncza (2006, 2010). In the CP literature more generally however, it is a well-known phenomena that symmetry breaking does not always improve performance as the ordering can conflict with the search strategy (Kiziltan, 2004). To understand when the symmetry breaking constraints would expect to improve performance, further experiments would be required.

CpOpt was the best method for improving the upper bounds. This is unsurprising as the default search in CpOpt has been specifically built to find good feasible solutions quickly for related scheduling problems. ROOSDDT was on average marginally the best method at improving the lower bounds. We will now further investigate these approaches on a known dataset in the literature.

3.6.2 Instances from MPSPlib

Experimental setup

The MPSPlib is a multi-project scheduling problem library for the BMPSP. The library consists of 140 instances, comprising of multi-project instances with 2, 5, 10, or 20 single-project instances. Each multi-project instance is made up of one or more RCPSP instances from PSPLIB of the same size. The number of global and local resources vary between instances as well as the arrival dates of the single projects and capacities of the resources. From the 140 instances, there exist 22 instances that contain project multiplicities but do not contain local resources. For all 22 of these instances each individual project consists of 90 activities and four global resources are available. Hence the number of activities vary from 180 to 1800 activities per

instance.

In some instances of the MPSPlib, individual projects have different arrival times. With respect to our notation let $a_{p,c}$ denote the arrival time of project $p \in P_c$ from class $c \in C$. All the models in this chapter can easily incorporate arrival times simply by modifying the project network used to determine the earliest and latest start times. More explicitly, the arc from the master dummy start node to the individual project start nodes is given the weight value of $a_{p,c}$. Moreover if we ensure that projects of the same class are defined in non-decreasing order of arrival times, the symmetry breaking methods hold.

The instances in MPSPLib are evaluated with respect to two different objective functions. The first is the Total Makespan (TMS), which is equivalent to the makespan as we have defined it in this chapter. The other objective is the Average Project Delay (APD), which is the average duration by which projects exceeds their Critical Path Duration (CPD). The CPD of a project is the makespan when an infinite amount of resources is available. Thus the APD is

$$APD = \frac{1}{n} \sum_{c \in C} \sum_{p \in P_c} \left(S_{\omega,p,c} - a_{p,c} - CPD_c \right) \tag{3.11}$$

In this section we consider both objectives explicitly. We run CpOpt both with and without the symmetry breaking constraints. We then use the ROOSDDT to see if the lower bound can be improved above the given resource-based lower bound and the bound given by CpOpt. CpOpt is run for 10 minutes per instances, but now with 4 threads and 2GB RAM. The ROOSDDT is given one hour, 8 threads and 8GB RAM. For the makespan objective the ROOSDDT is given the best known solution as the upper bound on the time-horizon. For APD objective determining an acceptable time-horizon is slightly more challenging as a schedule with the shortest TMS does not necessarily have the lowest APD. We use the objective function of the canonical schedule as the upper bound, which particularly for the larger instances can be seen as a conservative time-horizon.

No upper bound is given to CpOpt as the in-built heuristics are effective at finding good feasible solutions without having one provided.

Results / discussion

The results for the TMS objective are summarised in Table 3.4. Here *BKS* stands for the existing best known solution, *CP* is the solution obtained by CpOpt without symmetry breaking constraints, *CP-Sym* is the solution obtained by CpOpt with

Table 3.4: Summary of performance on relative instances from MPSPlib. A (*) is used to indicate optimal solutions, and a (†) indicates instances where the MIP could not solve the linear relaxation within the computational limits. OOM denotes out-of-memory.

ID	Instance	Upper Bound			Lower Bound		
		BKS	CP	CP-Sym	LB_R	LB_{CP}	LB_{MIP}
62	mp_j90_a10_nr5_AC10	172	171	**170**	166	168	**169**
66	mp_j90_a10_nr5_AC5	361	356	**355**	344	**347**	**347**
72	mp_j90_a20_nr5_AC10	483	479	**478**	469	471	**472**
80	mp_j90_a20_nr5_AC9	426	428	**414**	392	**392**	**392**
96	mp_j90_a5_nr5_AC5	256	250	**248**	241	244	**245**
71	mp_j90_a20_nr5_AC1	446	**436**	**436**	423	138	OOM†
76	mp_j90_a20_nr5_AC5	123	**119**	**119**	107	**118**	**118**
77	mp_j90_a20_nr5_AC6	236	**229**	**229**	**227**	96	**227**
78	mp_j90_a20_nr5_AC7	270	**266**	**266**	**265**	94	264†
79	mp_j90_a20_nr5_AC8	157	**145***	**145***	**145**	**145**	**145**
74	mp_j90_a20_nr5_AC3	159	**144***	**144***	128	**144**	**144**
82	mp_j90_a2_nr5_AC10	105	**101**	**101**	97	99	**100**
92	mp_j90_a5_nr5_AC10	254	**246**	**246**	241	**243**	**243**
90	mp_j90_a2_nr5_AC9	**329**	332	**329**	**275**	**275**	**275**
86	mp_j90_a2_nr5_AC5	**72***	**72***	**72***	**72**	**72**	**72**
73	mp_j90_a20_nr5_AC2	**127***	**127***	**127***	**127**	**127**	**127**
75	mp_j90_a20_nr5_AC4	363	**354**	355	323	328	**330**
65	mp_j90_a10_nr5_AC4	**667**	669	671	610	**613**	610†
70	mp_j90_a10_nr5_AC9	**233**	234	235	212	213	**215**
95	mp_j90_a5_nr5_AC4	**816**	835	824	686	686	**687**
100	mp_j90_a5_nr5_AC9	**808**	842	820	**686**	**686**	**686**
85	mp_j90_a2_nr5_AC4	**329**	**329**	330	275	275	**276**

symmetry breaking constraints, LB_R is the resource based lower bound, LB_{CP} is the lower bound obtained by CpOpt, and finally LB_{MIP} is the bound obtained by the ROOSDDT. The lower bounds of CpOpt with and without the symmetry breaking constraints were equivalent and hence only one column is shown, LB_{CP}. Furthermore the ROOSDDT does not improve the solution from the upper bound for any of the instances and hence only the lower bound is given in the graph.

Out of the 22 instances, 14 new best solutions were obtained and a solution with the same objective as the best known solution was obtained for a further 5 instances. Hence there were 4 instances for which neither CP or CP-Sym could find the best known solution or better. Out of the 14 new best solutions, 5 were obtained by just CP-Sym, 1 was obtained from just CP, and 8 were obtained by both CP and CP-Sym. Instances 74 and 79 were closed by both CP and CP-Sym.

The ROOSDDT was able to improve the lower bounds obtained by CpOpt and the resource lower bounds in 8, and obtain the equal best lower bound in a further 19 out of the relevant 22 instances. In the three instances where ROOSDDT does not at least equal the other two lower bounds (71, 78, 65) the solver was not able to solve the root relaxation within the specified time-limit or before running out of memory, i.e., exceeding 8GB RAM.

The lower bounds obtained by CP Optimizer on these instances is typically bounded below by LB_R and bounded above by LB_{CP}. CP Optimizer determines lower bounds as follows. An an initial lower bound is obtained by the minimum value of the objective variables given by propagation at the root note, which for the RCPSP/max is largely based on propagating the precedence constraints. Once the engine has found a number of solutions it then tries to compute a second lower bound by performing a dichotomy on the objective values with strong propagation algorithms. Finally during search the lower bound can be improved by Failure Directed Search (Vilim et al., 2015) through learning no-goods. The poor performance of LB_{CP} on instances 71, 77 and 78 is a result of CP Optimizer not being able to improve the initial lower bound by using the strong propagation algorithms within the time-limit.

The results for the APD objective are summarised in Table 3.5. The columns are the same as defined for Table 3.4. The MIP model used to obtain the lower bound is for this objective function is a slightly modified version of ROOSDDT. The objective function (3.7a) is replaced with the following objective function,

$$\min \quad \frac{1}{n} \sum_{c \in C} \Big(\sum_{t \in H} t(z_{\omega,c,t} - z_{\omega,c,t-1}) - \sum_{p \in P_c} a_{c,p} - m_c \cdot CPD_c \Big) \tag{3.12}$$

Furthermore as the master dummy-end activity is no longer required the precedence constraints (3.7e) and (3.7f) as well as the resource-based tightening constraints are not required.

Out of the 22 instances, 11 new best solutions were determined and a solution with the same objective as the best known solution was obtained for a further 2 instances. Out of the 11 new best solutions, 6 were obtained by CP-Sym, 4 were obtained by CP, and 1 was obtained by both CP and CP-Sym. No instances were closed for this objective function. It is also worth directly comparing the performance of CP and CP-Sym. Out of the 22 instances, CP-Sym outperforms CP on 14 instances, CP outperforms CP-Sym on 5 instances, and they obtain the same results on the remaining 3 instances. Although this is not conclusive it suggests that the

Table 3.5: Summary of performance on relative instances from MPSPlib for the Average Project Delay objective. OOM denotes out-of-memory.

ID	Instance	Upper Bound			Lower Bound	
		BKS	CP	CP-Sym	LB_{CP}	LB_{MIP}
62	mp_j90_a10_nr5_AC10	51.2	50.8	**50.2**	10.1	**46.1**
78	mp_j90_a20_nr5_AC7	77.6	70.9	**70.6**	0.2	OOM
79	mp_j90_a20_nr5_AC8	23.5	22.4	**20.1**	2.6	OOM
74	mp_j90_a20_nr5_AC3	10.2	8.50	**8.45**	0.9	**7.0**
82	mp_j90_a2_nr5_AC10	21.0	20.5	**20.0**	16.0	**17.0**
75	mp_j90_a20_nr5_AC4	116.9	116.7	**115.3**	0.3	**107.0**
72	mp_j90_a20_nr5_AC10	210.6	**208.8**	**208.8**	20.20	**195.0**
86	mp_j90_a2_nr5_AC5	**0.0**	**0.0**	**0.0**	0.0	0.0
73	mp_j90_a20_nr5_AC2	**0.0**	**0.0**	**0.0**	0.0	0.0
77	mp_j90_a20_nr5_AC6	62.9	**56.4**	58.4	0.2	OOM
80	mp_j90_a20_nr5_AC9	171.7	**169.5**	171.1	0.5	**158.3**
71	mp_j90_a20_nr5_AC1	139.8	**136.1**	138.6	0.3	OOM
76	mp_j90_a20_nr5_AC5	7.6	**5.7**	5.8	0.6	**5.3**
66	mp_j90_a10_nr5_AC5	**125.8**	130.7	128.0	23.6	**112.5**
96	mp_j90_a5_nr5_AC5	**88.8**	98.0	96.0	31.4	**73.8**
92	mp_j90_a5_nr5_AC10	**98.4**	102.4	101.2	35.2	**82.4**
90	mp_j90_a2_nr5_AC9	**179.0**	184.5	180.0	100.5	**102.0**
65	mp_j90_a10_nr5_AC4	**290.8**	304.3	297.7	49.7	**246.1**
70	mp_j90_a10_nr5_AC9	**79.1**	80.9	79.8	13.9	**71.8**
95	mp_j90_a5_nr5_AC4	**418.6**	461.0	446.8	119.2	**294.2**
100	mp_j90_a5_nr5_AC9	**428.2**	448.0	448.6	122.4	**304.2**
85	mp_j90_a2_nr5_AC4	**175.0**	186.0	183.0	**100.0**	**100.0**

symmetry breaking constraints are helping direct the search.

The ROOSDDT was able to considerably improve the lower bounds obtained by CpOpt when it did not run out of memory. The improvements in lower bound are very significant. For example consider instance ID 80, where the LB_{MIP} is 158.25 compared with LB_{CP} of 0.45.

3.7 Conclusion

In this chapter we have proven the existence of symmetry between projects of the same class in the high-multiplicity RCPSP/max. We explore two symmetry breaking approaches; (1) by adding symmetry breaking constraints in the form of precedences between activities of the same index from projects of the same class,

and (2) remodelling the problem to reduce the number of variables required. Our computational experiments show that both methods allow a number of different models to significantly increase the number of instances proven to optimality. The reduced version of the MIP models were found to outperform the expanded version of the MIP models with symmetry breaking constraints. The proposed MIP models which combine step variables and on-off variables also consistently outperform the pulse-start variable models.

There are a number of directions for future work. Firstly decomposition-methods have been shown to work effectively for multi-project scheduling (Toffolo et al., 2016). Given the smaller size of the reduced MIP models presented in this chapter, they should be useful in similar decomposition-based methods. The strong performance of Chuffed and Cpx with the symmetry breaking constraints in terms of proving optimality suggest these SAT-style CP solvers could be very useful in a destructive approach at obtaining lower bounds. This could be an effective comparison with the ROOSDDT model.

In general, the symmetry considered in this chapter does not exist for the multi-mode RCPSP. It is possible to find trivial counter examples where the symmetry does not hold in hybrid flow shop scheduling, which is well known to ba a special case of the multi-mode RCPSP (as considered by, e.g., Voß and Witt (2007)). Given the advantages we have demonstrated for the single-mode scheduling problem studied in this chapter we believe further study into symmetry of more general problems and their special cases is a topic that deserves further investigation.

CHAPTER 4

Liquid Handling Robot Scheduling Problem

4.1 Introduction

This chapter considers the Liquid Handling Robot Scheduling Problem (LHRSP), which seeks to minimise the time taken by a robot to transfer chemicals to a set of slides using a single pipette with finite capacity. The LHRSP is a sub-problem of the motivating real-world problem considered in Chapter 6 of this thesis. We study the LHRSP for two reasons. Firstly, we wish to understand how to model and solve this aspect of the real-world problem as even on its own it is a challenging scheduling problem. Secondly, we wish to develop a range of different approaches for the problem and empirically evaluate their performance. In this chapter we consider three different CP Optimizer models.

This work considerably extends results in the conference paper (Edwards et al. (2018)). The paper contains a MIP model, the first CP Optimizer model, and adapts two heuristics from the literature to obtain solutions to a restricted version of the problem. Due to their poor performance the heuristics have been omitted from this chapter. The MIP model can be found in Appendix B.1. The two improved CP Optimizer models appear here for the first time. The results presented in this chapter clearly reflect the benefits that these unpublished extensions make.

Perhaps the most significant insight gained in this chapter is that it is possible to obtain complete solutions to the problem by only searching on a subset of the variables. More specifically, if after a complete assignment of a certain subset of variables is made and constraint propagation reaches a fixed point, then it is possible to prove that the partial assignment can always be extended to a complete solution in a specific way. This insight results in considerable performance improvement and is the basis of the following two chapters.

4.1.1 Problem Description

The problem considers a single robot that is responsible for processing a set of jobs. In reality these jobs represent experiments that are processed through the sequential application of chemicals. When processing a single experiment the robot remains idle for the majority of the time, waiting for the correct chemical reactions to occur. Thus to make better use of its time, the robot is capable of processing many experiments in parallel.

The robot has a single pipette with finite volume that it uses to transfer chemicals to the set of n jobs $J := \{1, ..., n\}$. A simple schematic of the problem is given in Figure 4.1. The n jobs are arranged in the system according to their index, i.e., job $j \in J$ is at location j. The system also contains a set of vial containing different chemicals. These vials are located at location 0, as shown. To transfer a given chemical from a vial to job $j \in J$ the robot must (1) move to the vials, (2) *aspirate* chemical up into the pipette tip, (3) travel to job j, (4) *dispense* the chemical onto the sample. The robot takes $p^{\uparrow}$ time units to aspirate, and $p^{\downarrow}$ to dispense. The time taken for the robot to move from the location of job j to the location of job j' is denoted $p^{\rightarrow}_{j,j'}$, where $j, j' \in J \cup \{0\}$ and 0 represents the location of the vials.

Job $j \in J$ has N_j required operations, each of which corresponds to the dispensing of 1-unit of a certain chemical. To complete a job, chemicals are reqchapuired in a given order. Each job can be represented by a chain, as shown in Figure 2, where arcs represent the precedence relation between consecutive operations and nodes are color-coded to show the chemical required for the operation. Let C represent the set of the chemicals. Let $c_{i,j} \in C$ be the chemical required by operation i from job j. Note that one type of chemical might be required multiple times in the same job.

To ensure that the correct chemical reactions occur there exist minimum and maximum time lags between start times of consecutive operations from the same job. Let $O_j := \{1, 2, ..., N_j\}$ be all the operations of job $j \in J$. The time lags are

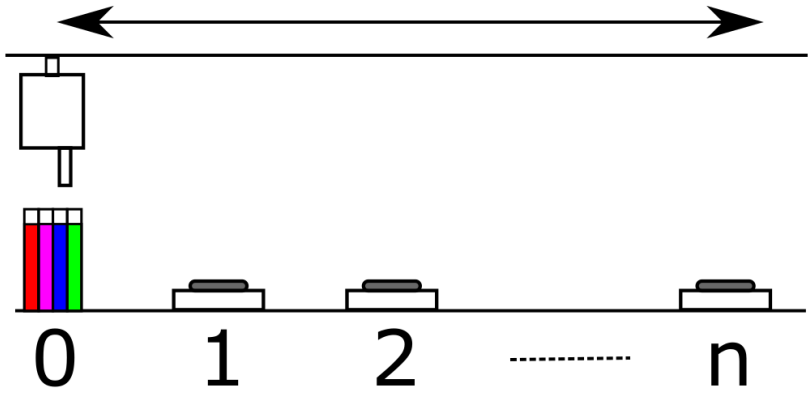


Figure 4.1: Schematic of the problem set-up. Vials are at location 0, jobs are at locations corresponding to the job's index. The robot can move between the vials and the jobs to transfer chemicals to the jobs.

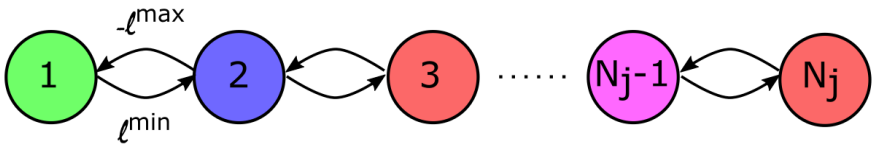


Figure 4.2: An example of a precedence graph that outlines the order in which the chemicals must be considered for a single job $j \in J$. Colours are used to indicate the chemical that is required by the operation. Positive and negative arc weights arise from the minimum and maximum timelags, respectively.

defined for all operations of job $j \in J$, except for the very last operation, i.e., $\{O_j \setminus \{N_j\}\}$. Hence once an operation, $i \in O_j \setminus \{N_j\}$ of job $j \in J$, has started there is a minimum amount of time units, $\ell_{i,j}^{min}$, before operation $(i+1)$ is allowed to start. Likewise once operation $i \in O_j \setminus \{N_j\}$ has started there is a maximum amount of time, $\ell_{i,j}^{max}$, within which operation $(i+1)$ must start. These time lags are expressed as

$$\ell_{i,j}^{max} \geq S_{i+1,j} - S_{i,j} \geq \ell_{i,j}^{min}, \quad (i \in O_j \setminus \{N_j\}, j \in J)$$

where $S_{i,j}$ is the start time of operation $i \in O_j$ from job $j \in J$,

The robot is allowed to aspirate multiple units of a single type of chemical at a time. This means the robot can, for example, (1) aspirate two units of a certain chemical, (2) move to a given job, (3) dispense one unit of the chemical at that job, (4) move to another job requiring the same chemical, (5) dispense the other unit of chemical at that job. For chemical $c \in C$ the robot can aspirate up to L_c units at a time. The time taken to aspirate multiple units of chemical is negligible and the processing time of each aspirate regardless of the quantity is $p^{\uparrow}$. Only one type of chemical can be in the pipette at a time. Given the robot can aspirate multiple units

of a chemical at a time, it can effectively reduce the number of times that it must aspirate the chemicals.

In practice there are typically more jobs than the system can accommodate at once, which often makes the system the bottleneck in other laboratory workflows. Hence the objective is to process all the jobs in the least amount of time, i.e., minimise makespan. We consider a job to be processed at the end time of the dispense of the final operation.

For simplicity, we assume in this chapter that once a job is completed it remains in the system. In reality though, as will be discussed in later chapters, once a job is completed it is removed from the system such that additional jobs can be inserted. Thus for now, we consider the problem as a static scheduling problem. Furthermore, the LHRSP problem assumes that the location of each job is fixed. Determining where each job should be located is an interesting and hard problem itself that can have significant impact on the quality of the schedule. The assignment of jobs to locations is considered in greater detail in Chapter 6. In this chapter however, no consideration is given to job location at all.

4.1.2 Related Problems

The LHRSP has a number of closely related problems studied in the literature. Firstly, to prove the complexity of the problem we consider the minimal-makespan single machine scheduling problem with unit processing times and minimum time lags (1p-SMSPmin), denoted by $PS1|chains(l_{ij}); p_i = 1|C_{max}$ using the project scheduling classification scheme proposed by Brucker et al. (1999a). The 1p-SMSPmin is known to be NP-Hard (Yu et al., 2004).

Theorem 4.1.1. *The LHRSP is NP-Hard*

Proof. We reduce 1p-SMSPmin to the LHRSP. Let I be an instance of 1p-SMSPmin. For each chain in I construct a job $j \in J$, where each activity in the chain corresponds to an activity $i \in O_j$. Let $\ell_{i,j}^{min} = l_{i,j}$ and $\ell_{i,j}^{max} = M$, where M is some sufficiently large number and $l_{i,j}$ is the minimum time lags specified in I. Now let $p^{\downarrow} = 1$, $p^{\uparrow} = 0$, $p_{j,j'}^{\rightarrow} = 0$, for all $j, j' \in J \cup \{0\}$. For completeness let $|C| = 1$, $c_{i,j} = c$ and $L_c = |\bigcup_{j \in J} O_j|$. As we can reduce 1p-SMSPmin to the LHRSP, and the 1p-SMSPmin is NP-Hard, then the LHRSP is also NP-Hard. □

If it is further assumed that only one unit of chemical can be transferred at a time, i.e., $c \in C$ is $L_c = 1$, the LHRSP can be simplified into a special case of the single machine scheduling problem with time lags (SMSP-TL), which is denoted

$PS1|chains, temp|C_{max}$, which is known to still be NP-Hard (Wikum et al., 1994). The aspirate, travel and dispense times are incorporated into the processing time of the activity. Hence all activities $i \in O_j$ from job j have the same processing time, defined by $p_j := p^{\uparrow} + p^{\rightarrow}_{0,j} + p^{\rightarrow}_{j,0} + p^{\downarrow}$. The time lags are still valid as all operations in the same job have the same processing time, now p_j as opposed to simply the dispense time $p^{\downarrow}$, and time lags only exist between operations of the same job.

The SMSP-TL is closely related to the well-studied job shop scheduling problem with time lags (JSP-TL), denoted by $J_m|temp|C_{max}$ (Brucker et al., 1999b). As pointed out in Caumond et al. (2008), when considering maximum time lags building non-trivial feasible solutions is not straightforward, however scheduling all operations of a single job after all operations of another job leads to feasible schedules. These types of schedules are referred to as *canonical schedules*. It is possible to generate similar feasible solutions to the LHRSP, albeit very poor ones, by completing all operations of one job, then all operations of the next job, and so on.

To find better initial solutions than the canonical schedules for the JSP-TL, Caumond et al. (2008) proposed a list-based heuristic combined with a method for repairing solutions. Following this, Artigues et al. (2011) proposed a job insertion heuristic (JIH), which exploits the fact that precedences only exist between consecutive operations of the same job. In the experimental study in Artigues et al. (2011), the JIH is shown to produce better solutions than the list-based heuristic from Caumond et al. (2008). Moreover the state-of-the-art approach to the JSP-TL (González et al., 2015) also utilises the JIH to obtain initial feasible solutions to their scatter-search with path-relinking approach. In the conference paper (Edwards et al., 2018), we adapted the JIH to the LHRSP, however it has been omitted from this thesis due to its relatively poor performance.

Both the SMSP-TL and JSP-TL are special cases of the well-studied resource-constrained project scheduling problem with generalised precedence constraints (RCPSP-max), denoted by $P|temp|C_{max}$. CP (CP) approaches, such as Schutt et al. (2013c) and Vilim et al. (2015), have been particularly effective for this RCPSP-max. Furthermore for the RCPSP-max there are a number of effective schedule generation schemes such as serial schedule generation scheme (SSGS) with unscheduling step, also referred to as the *direct method* proposed by Franck et al. (2001). This method has been modified for many practical applications, such as the generalised surgery scheduling problem (Riise et al., 2016). In the conference paper (Edwards et al., 2018), we adapted the SSGS to the LHRSP, however it has also been omitted from this thesis due to its relatively poor performance.

The LHRSP also closely relates to a number of scheduling problems where material handling is considered. In particular, the problem has a strong resemblance to hoist scheduling problems (HSP)s, which have been motivated by electroplating lines. There are many variants of the HSP considered in the literature, see Manier and Bloch (2003) for a very detailed classification. In general one or more hoists move different *carriers*, which can be thought of as different jobs, between chemical baths, within which the jobs must remain between a minimum and maximum allowable time. Superficially, the LHRSP resembles a type of HSP where instead of transferring jobs between different chemical baths, the hoist transfers different chemicals to the jobs.

From the perspective of scheduling, the LHRSP has a number of additional challenges that make it difficult to model as a type of HSP. In HSPs, a hoist carries a single job at a time. Furthermore once a job has been lifted from one chemical bath, the hoist must travel and lower the job into the next chemical bath immediately. This is known as the *no-wait requirement* and is enforced to ensure the jobs are not damaged by oxidation during transfer. A result of this is that the transport tasks have a known, fixed duration, and the scheduling of the time the jobs are in the baths can be accounted for in the scheduling of the transfer tasks. In contrast, the LHRSP can transfer a number of units of chemicals simultaneously. This introduces a number of fundamental differences, such as not knowing how many times the robot must aspirate in a solution.

Finally, the problem can be seen as a travelling salesman problem (TSP) with minimum and maximum time lags with additional side constraints. To the best of our knowledge, no papers have studied the pure TSP with minimum and maximum time lags between deliveries, however similar problems such as the vehicle routing problem with temporal dependencies have been studied by Dohn et al. (2011). Practical applications that consider vehicle routing with time lags include homecare crew routing problem (Rasmussen et al., 2012) and the concrete delivery problem (Kinable et al., 2014), for which both CP and MIP models have been effective.

4.2 CP Optimizer Models

In this section we propose three different CP Optimizer models. A MIP model for the LHRSP can be found in Appendix B.1. It is not included in the body of the thesis due to its relatively poor performance in practice. Furthermore, let $O = \bigcup_{j \in J} O_j$ be

the set of all operations and to distinguish between dispenses of different chemicals, let all the activities requiring $c \in C$ be defined as $O^c = \{(i,j) \in O | c_{i,j} = c\}$.

4.2.1 Model 1 - (CP1)

Overview

The first CP Optimizer model, CP1, was introduced in the conference paper (Edwards et al., 2018). The model is based on the observation that the liquid handling robot iterates between aspirating a chemical and carrying the chemical to dispense on the different jobs. The optimal number of times aspiration needs to be completed is not known before scheduling, however the maximum number should never exceed the total number of dispenses, i.e., aspirate one unit of chemical for each dispense. An important distinction of this model compared with the subsequent models is that in CP1 we do not consider aspirating as part of a job.

The model considers four types of interval variables all of which are compulsory. Firstly, a dispense variable, $\alpha_{i,j}^{dispense}$, is defined for each operation $(i,j) \in O$ and represents the interval in which chemical is being dispensed with duration equal to $p^{\downarrow}$. Secondly, an aspirate variable, $\alpha_q^{aspirate}$, is defined for (but not associated with) each possible operation $q \in [1, ..., |O|]$ and represents the interval in which chemical is being aspirated with duration equal to $p^{\uparrow}$. For clarity, we emphasise that the aspirate intervals are defined independent of specific dispense operations. Precedence constraints re added to enforce an ordering on the aspirate intervals such that, for example, $\alpha_1^{aspirate}$ will occur first, $\alpha_2^{aspirate}$ will occur second and so on. Clearly, it is not possible to enforce such an ordering on the dispense intervals. It is also worth stating that although all of the aspirate variables are defined to be compulsory, the objective function will be defined in such a way that unused aspirate intervals can be removed in post-processing and does not violate the validity of the model. The third type of interval variable, is a carry variable, α_q^{carry}, that represents an interval where chemical is being carried in the pipette is defined for each possible operation $q \in [1, ..., |O|]$ with a minimum length of $p^{\downarrow}$. Fourthly, a cover variable, $\alpha_{i,j}^{cover}$, that represents an interval that spans the associated dispense interval is defined for each operation $(i,j) \in O$ also with a minimum length of $p^{\downarrow}$. Constraints will be added such that the dispense variables are dispersed correctly between the aspirate variables by coordinating the carry and cover intervals.

To ensure the correct behaviour occurs between the interval variables a number of sequence variables, state functions and cumulative functions are defined.

Firstly, to ensure the travel times are respected a sequence variable, $\rho^{location}$, is defined over the aspirate and dispense variables, where the types of the variables are associated with the location where the variables must occur. Hence the type associated of the aspirate variables are all set to 0, the location of the vials, whereas the type associated with the dispense variables are set to the index of the corresponding job j. Secondly, to ensure the correct transfer of chemicals two state functions are defined. The carry state function, ψ^{carry}, is 1 when chemical is being carried and 0 otherwise. The pipette state function, $\psi^{pipette}$, is always equal to the index of the chemical $c \in C$ currently being carried to avoid cross contamination occuring. Finally, for each chemical $k \in R$ a cumul function, f^k, is used to ensure the amount of chemical does not exceed the limit of the pipette. More explicitly, the carry variables will increase the cumul functions by the maximum number of units that can be carried of each chemical. The cover intervals will decrease the appropriate cumul function by 1 unit for their duration.

Intuition

The intuition of CP1 is visualised in Figure 4.3. Here we consider a subset of three operations from a possible schedule, (i, j), (i', j'), (i'', j''), where two operations require the chemical associated with the colour pink, $c_{i,j} = c_{i',j'} = c_1$, $L_1 = 2$, the other requires the chemical associated with the colour blue, $c_{i'',j''} = c_2$, $L_2 = 1$. The dispense and cover variables are indexed by their associated operations. Whereas the aspirate and carry variables are indexed chronologically. An ordering is forced on the aspirate and carry variables such that they alternate and lower indexed variables are sequenced earlier than higher indexed variables. Cover variables start before the start and end after the end of the corresponding dispense variables. The carry state ensures that when a cover and carry interval overlap they start together and end together.

Each carry variable produces the maximum number of units possible for each of the chemicals, hence during the first carry variable, $carry\_var_1$, 1 unit of the blue unit and 2 pink units are produced corresponding to the maximum number of units that the pipette can carry at a time for each chemical. The cover variables of (i, j) and (i', j') consume 1 unit each of the pink chemical for their duration, hence the amount produced by the carry variable is cancelled out by the cover variables. Whereas the one blue unit produced by the carry variable is neither consumed by either (i, j) nor (i', j') hence there is a net increase in the amount of blue chemical of 1 unit during this interval. Here we emphasise that it is possible multiple dispense

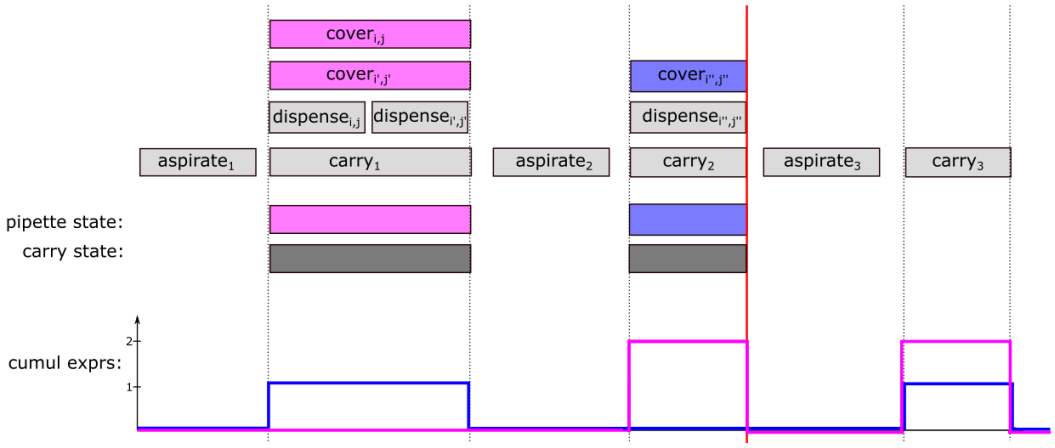


Figure 4.3: Visual summary of CP1

intervals during a carry interval. The colour state is used to ensure that only all pink cover variables or all blue cover variables overlap and hence prevent cross contamination.

The makespan is taken as the end of time of the last dispense, indicated here with the red line. Note that the third aspirate and carry intervals are scheduled after the makespan. These remaining interval can be simply removed in a post-processing stage.

Model

The first CP Optimizer model, CP1, can now be defined as follows.

$$min \quad \max_{(i,j)\in O} \texttt{endOf}(\alpha_{i,j}^{dispense}) \tag{4.1a}$$

$$s.t. \quad \texttt{startOf}(\alpha_1^{aspirate}) = 0 \tag{4.1b}$$

$$\texttt{endBeforeStart}(\alpha_q^{aspirate}, \alpha_q^{carry}) \qquad \forall q \in [1, ..., |O|] \tag{4.1c}$$

$$\texttt{endBeforeStart}(\alpha_q^{carry}, \alpha_{q+1}^{carry}) \qquad \forall q \in [1, ..., |O| - 1] \tag{4.1d}$$

$$\texttt{startBeforeStart}(\alpha_{i,j}^{dispense}, \alpha_{i+1,j}^{dispense}, \delta_{i,j}^{min}) \qquad \forall i \in O_j; j \in J \tag{4.1e}$$

$$\texttt{startBeforeStart}(\alpha_{i+1,j}^{dispense}, \alpha_{i,j}^{dispense}, -\delta_{i,j}^{max}) \qquad \forall i \in O_j; j \in J \tag{4.1f}$$

$$\texttt{startBeforeStart}(\alpha_{i,j}^{cover}, \alpha_{i,j}^{dispense}) \qquad \forall (i,j) \in O \tag{4.1g}$$

$$\texttt{endBeforeEnd}(\alpha_{i,j}^{dispense}, \alpha_{i,j}^{cover}) \qquad \forall (i,j) \in O \tag{4.1h}$$

$$\texttt{alwaysEqual}(\psi^{pipette}, \alpha_{i,j}^{cover}, c_{i,j}, \leftrightarrow) \qquad \forall (i,j) \in O \tag{4.1i}$$

$$\texttt{alwaysEqual}(\psi^{carry}, \alpha_{i,j}^{cover}, 1, \leftrightarrow) \qquad \forall (i,j) \in O \tag{4.1j}$$

$$\texttt{alwaysEqual}(\psi^{carry}, \alpha_q^{carry}, 1, \leftrightarrow) \qquad \forall q \in [1, ..., |O|] \tag{4.1k}$$

$$f_k^{cap} \mathrel{+}= \texttt{pulse}(\alpha_q^{carry}, L_c)) \qquad \forall k \in R; (i,j) \in O : c_{i,j} = c \qquad (4.1l)$$

$$f_k^{cap} \mathrel{-}= \texttt{pulse}(\alpha_{i,j}^{cover}, 1)) \qquad \forall k \in R; (i,j) \in O : c_{i,j} \neq c \qquad (4.1m)$$

$$f_k^{cap} \geq 0 \qquad \forall k \in R \qquad (4.1n)$$

$$\texttt{inSequence}(\rho, \alpha_{i,j}^{dispense}, j) \qquad \forall (i,j) \in O \qquad (4.1o)$$

$$\texttt{inSequence}(\rho, \alpha_q^{aspirate}, 0) \qquad \forall (i,j) \in O \qquad (4.1p)$$

$$\texttt{noOverlap}(\rho, M) \qquad (4.1q)$$

$$\alpha_{i,j}^{dispense} \in \texttt{interval}(comp, p^{\downarrow}) \qquad \forall (i,j) \in O \qquad (4.1r)$$

$$\alpha_{i,j}^{cover} \in \texttt{interval}(comp, (p^{\downarrow}, \infty)) \qquad \forall (i,j) \in O \qquad (4.1s)$$

$$\alpha_q^{aspirate} \in \texttt{interval}(comp, p^{\uparrow}) \qquad \forall q \in [1, ..., |O|] \qquad (4.1t)$$

$$\alpha_q^{carry} \in \texttt{interval}(comp, (p^{\downarrow}, \infty)) \qquad \forall q \in [1, ..., |O|] \qquad (4.1u)$$

$$f_k \in \texttt{cumul} \qquad \forall k \in R \qquad (4.1v)$$

$$\psi^{pipette}, \psi^{carry} \in \texttt{state} \qquad (4.1w)$$

$$\rho \in \texttt{sequence} \qquad (4.1x)$$

The objective (4.1a) to minimise the completion time of the last dispense variables. Constraint (4.1b) fixes the first aspirate to start at the beginning of the schedule. Constraints (4.1c) and (4.1d) then ensure that the aspirate variables alternate with the carry variables. Constraints (4.1e) and (4.1f) ensure the minimum and maximum time lags, respectively, are respected between consecutive operations from the same job. Constraints (4.1g) and (4.1h) ensure the cover variables start before the start, and end after the end of the corresponding dispense variable, respectively. Together, constraints (4.1i)-(4.1k) ensure that only a single chemical can be carried by the pipette at a time. Constraint (4.1i) ensures that the interval of the pipette state must always be equal to the corresponding value of the chemical required by the cover variables, and these intervals are both left and right aligned. Likewise, constraints (4.1j) and (4.1k) ensure that the interval of the carry state must always be equal to one during both the cover and carry intervals, and these intervals are both left and right aligned. Constraints on cumulative functions (4.1l)-(4.1n) then ensure the cover, and thus dispense variables, must only occur in parallel with a carry variable, and that the quantity of chemical carried does not exceed the capacity of the pipette. To do so, constraint (4.1l) ensures that each carry interval produces the maximum amount of units of chemical that be carried by the pipette for each chemical. Then constraint (4.1m) ensures that the cover intervals consume a single unit of the appropriate chemical, and constraint (4.1n) ensures the cumulative functions are never negative. Constraint(4.1q) ensures the correct travel times between the

aspirate and dispense variables, where $M = ((p^{\rightarrow}_{l,l'})_{l \in L})_{l' \in L}$ is a distance matrix between all the different locations. Finally, the interval variables, sequence variables, state and cumulative functions are defined by constraints (4.1r-4.1x).

Comments

There are a number of aspects of the model that can be improved. Two sets of interval variables; (1) the aspirate and carry variables and (2) the dispense and cover variables, are not strongly constrained. More explicitly, the nodes associated with the aspirate and carry variables in the inferred temporal network are completely disjoint from the remaining nodes. Thus the model relies on the cumul expressions and state functions to ensure overlapping carry and cover variables are synchronised to propagate. Strongly connected components of the temporal network are also important when building initial solutions. To find initial solutions CP Optimizer uses a topological sort in the temporal network to determine sets of activities to insert one at a time. The strongly connected components in the temporal network for CP1 are (1) all the dispense and cover variables for each job and (2) one chain of aspirate and carry variables. These strongly connected components are not particularly insightful into the structure of the problem as for example it is not possible to schedule the connected component of dispense and cover variables without also scheduling a section of the chain of aspirate and carry variables due to the interaction of the cumulative expression.

Also CP1 defines a different cumulative expression for each chemical. However due to the pipette state function, only one chemical can be consumed by a dispense at any point in time and thus only a single cumulative expression can be violated at any point in time. Hence it should be possible to model the problem using a single cumulative expression.

4.2.2 Model 2 - (CP2)

Overview

A second CP Optimizer model, CP2, is given that we would expect to in general outperform CP1. The key difference between CP1 and CP2 is that the latter explicitly indexes aspirate variables to specific operations, i.e. an aspirate variable $\alpha^{aspirate}_{i,j}$ is defined for each operation $(i, j) \in O$. To account for the fact that multiple units of chemical can be aspirated at once the aspirate variables are allowed to overlap in time. This contrasts to CP1 where aspirates are not constrained to specific

dispenses, are not allowed to overlap in time and potentially not all of the aspirate variables are used. The model CP2 now considers only two other types of interval variables. Dispense variables, $\alpha_{i,j}^{dispense}$, are defined identically to CP1. We refer to the final type of interval variable as *serve variables*. A serve variables, $\alpha_{i,j}^{serve}$, is defined with respect to each specific operation, $(i,j) \in O$, and represents the interval of time from the start of the corresponding aspirate to the end of the corresponding dispense. No restrictions are placed on the length of the serve variables.

A sequence variable, $\rho^{location}$, is defined over just the dispense variables, where the type of the variables are still associated with the index of the corresponding job. The aspirate variables are now not considered by the sequence variable as we allow them to overlap. Instead we define a location state function, $s^{location}$, where the value of the intervals represent the location of the robot and the distance matrix M ensures travel times are respected through these transitions. Again the pipette state, $\psi^{pipette}$, is always equal to the index of the chemical $c \in C$ currently being carried. Finally, a single cumulative function, $f^{capacity}$, is defined that represents the amount of chemical being carried by the pipette.

Intuition

The intuition of CP2 is visualised in Figure 4.4. Here we consider the equivalent CP2 schedule for the example used for CP1 in Figure 4.3. A significant difference is that the aspirate variables are now associated with specific dispenses. As can be seen, the serve variables start at the start of the corresponding aspirate and end at the end of the corresponding dispense. The pipette state then ensures that only serves corresponding to operations which require the same chemical are allowed to overlap. Furthermore a single cumul expression is used to keep track of how much chemical is being carried in the pipette. The cumul is constrained to always be in an acceptable range during the intervals of the serve variables, which is visualised by ensuring that the black line is always within the coloured rectangles. Finally the pos state keeps track of where the robot is, the values in the associated rectangles represent the type of the interval used to enforce the sequence-dependent setup times.

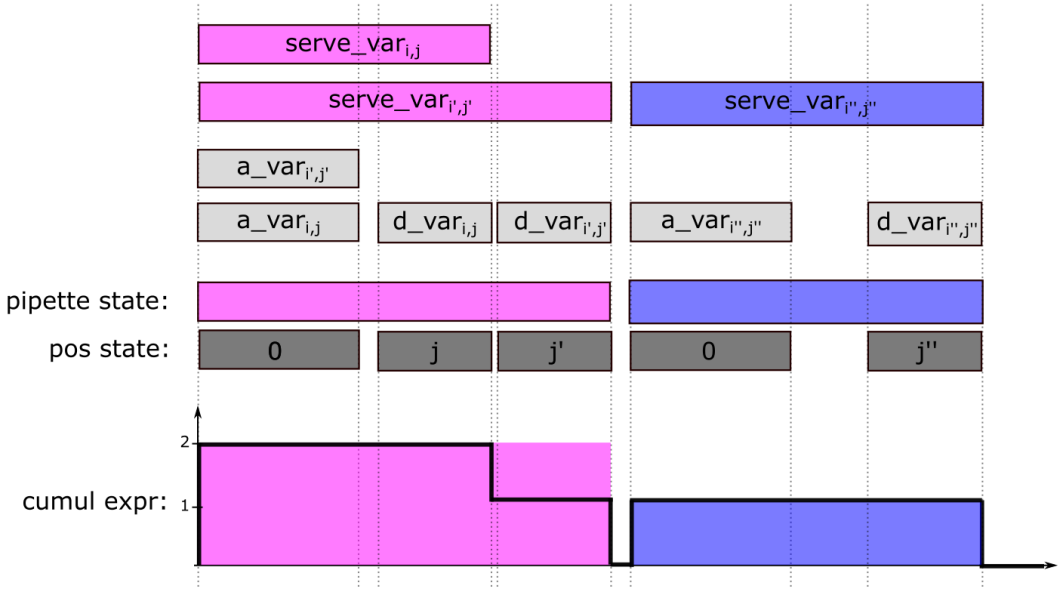


Figure 4.4: In CP2 an aspirate variable is associated with each aspirate.

Model

The second CP Optimizer model, CP2, can now be defined as follows.

$$min \quad \max_{(i,j)\in O} \texttt{endOf}(\alpha_{i,j}^{dispense}) \tag{4.2a}$$

$$s.t. \quad \texttt{startBeforeStart}(\alpha_{i,j}^{dispense}, \alpha_{i+1,j}^{dispense}, \delta_{i,j}^{min}) \qquad \forall i \in O_j; j \in J \tag{4.2b}$$

$$\texttt{startBeforeStart}(\alpha_{i+1,j}^{dispense}, \alpha_{i,j}^{dispense}, -\delta_{i,j}^{max}) \qquad \forall i \in O_j; j \in J \tag{4.2c}$$

$$\texttt{startAtStart}(\alpha_{i,j}^{aspirate}, \alpha_{i,j}^{serve}) \qquad \forall (i,j) \in O \tag{4.2d}$$

$$\texttt{endAtEnd}(\alpha_{i,j}^{dispense}, \alpha_{i,j}^{serve}) \qquad \forall (i,j) \in O \tag{4.2e}$$

$$\texttt{endBeforeStart}(\alpha_{i,j}^{aspirate}, \alpha_{i,j}^{dispense}, p_{0,j}^{\rightarrow}) \qquad \forall (i,j) \in O \tag{4.2f}$$

$$\texttt{alwaysEqual}(\psi^{pipette}, \alpha_{i,j}^{serve}, c_{i,j}, \leftarrow) \qquad \forall (i,j) \in O \tag{4.2g}$$

$$\texttt{alwaysEqual}(\psi^{location}, \alpha_{i,j}^{dispense}, j, \leftrightarrow) \qquad \forall (i,j) \in O \tag{4.2h}$$

$$\texttt{alwaysEqual}(\psi^{location}, \alpha_{i,j}^{aspirate}, 0, \leftrightarrow) \qquad \forall (i,j) \in O \tag{4.2i}$$

$$f^{capacity} \mathrel{+}= \texttt{pulse}(\alpha_{i,j}^{serve}, 1) \qquad (i,j) \in O \tag{4.2j}$$

$$\texttt{alwaysIn}(f^{capacity}, \alpha_{i,j}^{serve}, 0, L_{c_{i,j}}) \qquad (i,j) \in O \tag{4.2k}$$

$$\texttt{noOverlap}(\rho^{location}, M) \tag{4.2l}$$

$$\alpha_{i,j}^{dispense} \in \texttt{interval}(comp, p^{\downarrow}) \qquad \forall (i,j) \in O \tag{4.2m}$$

$$\alpha_{i,j}^{aspirate} \in \texttt{interval}(comp, p^{\uparrow}) \qquad \forall (i,j) \in O \tag{4.2n}$$

$$\alpha_{i,j}^{serve} \in \texttt{interval}(comp, p_{i,j}^{serve}) \qquad \forall (i,j) \in O \tag{4.2o}$$

$$f^{capacity} \in \texttt{cumul} \tag{4.2p}$$
$$\psi^{pipette} \in \texttt{state} \tag{4.2q}$$
$$\psi^{location} \in \texttt{state}(M) \tag{4.2r}$$
$$\rho^{location} \in \texttt{sequence}((\alpha_{i,j}^{dispense}, j)_{(i,j)\in O}) \tag{4.2s}$$

The objective (4.2a) is to minimise the maximum completion time of any operation. Constraints (4.2b) and (4.2c) represent the minimum and maximum time lags between dispense operations in the same job, respectively. Constraints (4.2d) and (4.2e) ensure the serve variables start at the start of the associated aspirate variable and end at the end of the associated dispense variable, respectively. Constraints (4.2f) ensure that the aspirate variables occurs before the associated dispense variables. Constraints (4.2g) ensure that the values of the intervals of the pipette state must always be equal to the corresponding value of the chemical required by the serve variables, and these intervals are left aligned. In doing so, this ensures that if two serve variables overlap, then their aspirate variables start at the same time. Constraints (4.2h) and (4.2i) ensure that the values of the intervals of the location state must always be equal to 0 for the aspirate variables and the appropriate job index j for the corresponding dispense variables, respectively. Constraints (4.2j) increases the cumulative expression by a single unit for the duration of each serve variable. Then constraints (4.2k) ensure that the value of the cumulative expression is always between 0 and the limit $L_{c_{i,j}}$ throughout the serve interval associated with each operation $(i,j) \in O$. Constraints (4.2l) ensure the location sequence does not overlap and has the correct travel times matrix M. Finally constraints (4.2m)-(4.2s) define the various decision variables and functions.

Comments

There are still a number of improvements which can be made to CP2. Firstly, it is possible to normalise the different pipette capacities of the different chemicals such that the capacity of the pipette is constant across the time-horizon. This will benefit powerful propagation algorithms that work on the global capacity of a resource, such as *edge-finder* (Vilím, 2009), which is used by CP Optimizer. In order to normalise the carrying capacities, we determine the lowest common multiple, LCM, of all the carrying capacities, i.e., L_c for all $c \in C$. Then for each dispense operation $(i,j) \in V^c$ requiring chemical $c \in C$ we associate an equivalent chemical consumption, $q_{i,j} := \frac{LCM}{L_c}$.

Secondly, a number of strengthening constraints can be added that will provide more information to the temporal network. Additional precedence constraints can be added between aspirates and dispenses from the same job. More explicitly, if two consecutive dispenses from the same job require different chemicals, then the corresponding aspirate intervals cannot overlap. Hence the preceding dispense must be completed before the following aspirate can begin, while also considering the transition time in between. Alternatively, if two consecutive dispenses require the same chemical, then it is possible for the two aspirates to occur simultaneously. However the succeeding aspirate cannot start before the preceding aspirate has started.

4.2.3 Model 3 - (CP3)

Overview

A third CP Optimizer model, CP3, is given that we would expect to outperform CP2. The key difference between CP2 and CP3 is that the latter does not model the aspirate intervals explicitly. The model considers only two types of interval variables. Again the dispense variables, $\alpha_{i,j}^{dispense}$, are defined identically to CP1 and CP2. The serve variables, $\alpha_{i,j}^{serve}$, are again defined and still represent the time from the start of the corresponding aspirate until the time after the end of the corresponding dispense required to return to the vials. To account for the fact that aspirate variables are not defined the length of the serve variables are defined to be at least $p_j^{serve} = p^{\uparrow} + p_{0,j}^{\rightarrow} + p^{\downarrow} + p_{j,0}^{\rightarrow}$. This assumes that the triangle inequality holds with the travel time. The model considers a location sequence, $\rho^{location}$, considering just the dispense variables, where again the type of the variable is set to the job index j of the operation. The model considers a single cumulative function, $f^{capacity}$, that represents the (normalised) quantity of chemical being carried by the pipette. The model considers a single state function, the pipette state $f^{pipette}$ where the intervals are always equal to the index of the chemical $c \in C$ being carried.

Intuition

The intuition of CP3 is visualised in Figure 4.5. Much of the intuition of the model of CP3 is the same as CP2 yet there are a number of key differences. The serve variables now start and end at the vials and thus include the time taken to travel back to the vials once a dispense is completed. The aspirate variables and the position state are no longer required. The cumulative expression has also been reformulated.

Now the cumulative expression is constrained to not exceed the capacity of the pipette across the entire schedule. During serve variables the amount of chemical in the pipette is increased. Note that during the transfer of the chemical for the two operations requiring the pink chemical, the pipette is increased to the capacity of the pipette at first, and then decreased to half of that after the end of the first serve variable. For the blue chemical, the cumulative expression is increased to the capacity of the pipette despite only one serve variable being completed during this time. This is because the pipette can only transfer one-unit of blue chemical at a time.

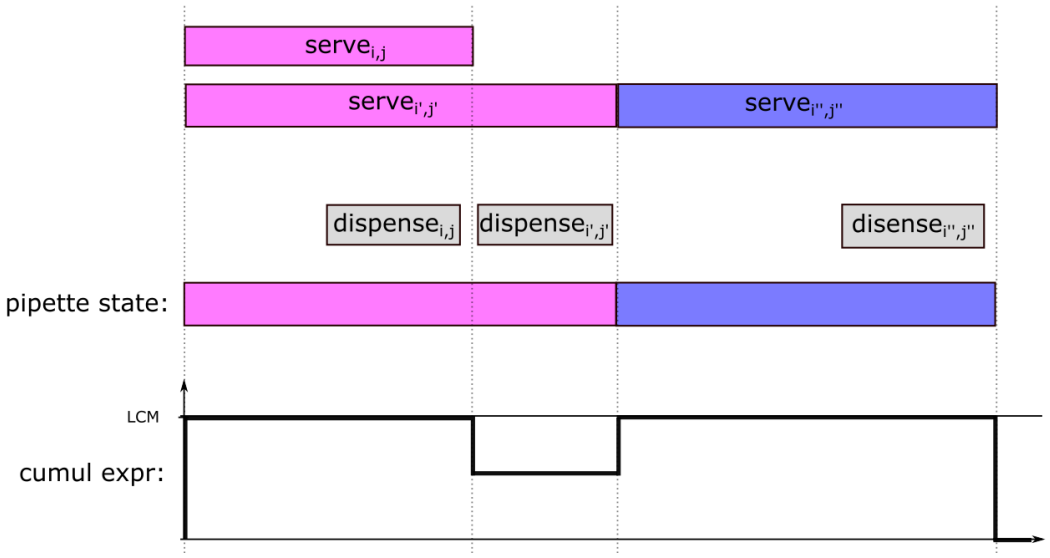


Figure 4.5: Visual summary of CP3

Model

The third CP Optimizer model, CP3, can now be defined as follows.

$$min \quad \max_{(i,j)\in O} \texttt{endOf}(\alpha_{i,j}^{dispense}) \tag{4.3a}$$

$$s.t. \quad \texttt{startBeforeStart}(\alpha_{i,j}^{dispense}, \alpha_{i+1,j}^{dispense}, \delta_{i,j}^{min}) \qquad \forall i \in O_j; j \in J \tag{4.3b}$$

$$\texttt{startBeforeStart}(\alpha_{i+1,j}^{dispense}, \alpha_{i,j}^{dispense}, -\delta_{i,j}^{max}) \qquad \forall i \in O_j; j \in J \tag{4.3c}$$

$$\texttt{endAtEnd}(\alpha_{i,j}^{dispense}, \alpha_{i,j}^{serve}, d_{j,0}^{\rightarrow}) \qquad \forall (i,j) \in O \tag{4.3d}$$

$$\texttt{startBeforeStart}(\alpha_{i,j}^{serve}, \alpha_{i+1,j}^{serve}) \qquad \forall (i,j) \in O : c_{i,j} = c_{i+1,j} \tag{4.3e}$$

$$\texttt{endBeforeStart}(\alpha_{i,j}^{serve}, \alpha_{i+1,j}^{serve}) \qquad \forall (i,j) \in O : c_{i,j} \neq c_{i+1,j} \tag{4.3f}$$

$$\texttt{alwaysEqual}(\psi^{pipette}, \alpha_{i,j}^{serve}, c_{i,j}, \leftarrow) \qquad \forall (i,j) \in O \tag{4.3g}$$

$$f^{cap} = \sum_{(i,j)\in O} \texttt{pulse}(\alpha^{serve}_{i,j}, q(c_{i,j})) \leq R^{pipette} \tag{4.3h}$$

$$\texttt{noOverlap}(\rho^{location}, M) \tag{4.3i}$$

$$\alpha^{dispense}_{i,j} \in \texttt{interval}(comp, p^{\downarrow}) \qquad \forall (i,j) \in O \tag{4.3j}$$

$$\alpha^{serve}_{i,j} \in \texttt{interval}(comp, (p^{serve}_{j}, \infty)) \qquad \forall (i,j) \in O \tag{4.3k}$$

$$f^{capacity} \in \texttt{cumul} \tag{4.3l}$$

$$\psi^{pipette} \in \texttt{state} \tag{4.3m}$$

$$\rho^{location} \in \texttt{sequence}((\alpha^{dispense}_{i,j}, j)_{(i,j)\in O}) \tag{4.3n}$$

The objective (4.3a) is to minimise the maximum completion time of the dispense variables. Constraints (4.3b) and (4.3c) enforce the minimum and maximum time lags respectively. Constraints (4.3d) ensure the end of the serve variables occurs exactly the amount of time required to return to the vials after the end of the corresponding dispense variable. Constraints (4.3e) strengthen the temporal network by ensuring that if consecutive operations of the same job require the same chemical then the start of the serve variable of the predecessor does not start after the start of the serve variable of the successor. Similarly, constraints (4.3f) ensure that if consecutive operations from the same job require different chemicals then the end of the serve variable associated with the predecessor occurs not after the start of the serve variable of the successor. Constraints (4.3g) ensure the value of the intervals of the pipette state must always be equal to the corresponding value of the chemical required by the serve variables, and these intervals are left aligned. Constraint (4.3h) ensures that the serve variable associated with each operation (i,j) increases the value of the cumulative function by the normalised quantity $q(c_{i,j})$ for the correct chemical, and that the value of the cumulative function never exceeds the normalised capacity $R^{pipette}$. Constraint (4.3i) ensures the location sequence does not overlap and respects the travel times. Finally, constraints (4.3j)-(4.3n) define the various decision variables and functions.

Search Phase Theory

Let A be an assignment over the set of dispense variables that is not disallowed by any constraints, i.e., the start times (and thus end times) of the dispense variables are fixed such that constraint propagation does not fail. In this section we prove that if an assignment A is made and constraint propagation does not fail then it is possible to construct a feasible solution.

Proposition 4.2.1. *Given A the minimum and maximum time lag constraints and the noOverlap constraint are satisfied.*

Proof. As the span of both the minimum and maximum time lag constraints, (4.3b) and (4.3c) respectively, and the noOverlap constraint, (4.3i), are only over the dispense variables, any assignment A over the dispense variables that is not disallowed by any constraints in the model must satisfy these constraints. □

From Proposition 4.2.1, the dispense variables in the assignment A are in a non-overlapping sequence, $seq(A)$. From constraint 4.3d, after propagation the ends of the serve variables are also fixed. Hence the only remaining unfixed variables are the start times of the serve variables. As we know that assignment A is not disallowed by any constraints, we know that after propagation the domains of these variables are non-empty, i.e., $\texttt{ES}(\alpha^{serve}_{i,j}) \leq \texttt{LS}(\alpha^{serve}_{i,j})$. Let us now consider the possible values for the latest start of the serve variables.

Proposition 4.2.2. *Given A, let (i,j) directly precede (i',j') in the sequence $seq(A)$, the latest start time of the latter serve variable after propagation is*

$$LS(\alpha^{serve}_{i',j'}) = \begin{cases} LS(\alpha^{serve}_{i,j}), & \text{if } S(\alpha^{dispense}_{i',j'}) - C(\alpha^{dispense}_{i,j}) < p^{\rightarrow}_{j,0} + p^{\uparrow} + p^{\rightarrow}_{j',0} \\ S(\alpha^{dispense}_{i',j'}) - p^{\uparrow} - p^{\rightarrow}_{0,j'}, & \text{otherwise} \end{cases} \tag{4.4}$$

Proof. Given assignment A after propagation of the *end-at-end* constraints and the minimum length of the serve variables, for each operation $(i,j) \in O$ we know that $\texttt{LS}(\alpha^{serve}_{i,j}) \leq \texttt{S}(\alpha^{dispense}_{i,j}) - p^{\uparrow} - p^{\rightarrow}_{0,j}$. Hence the proof reduces to proving that $\texttt{LS}(\alpha^{serve}_{i',j'})$ is further filtered to $\texttt{LS}(\alpha^{serve}_{i,j})$ if and only if $\texttt{S}(\alpha^{dispense}_{i',j'}) - \texttt{C}(\alpha^{dispense}_{i,j}) < p^{\rightarrow}_{j,0} + p^{\uparrow} + p^{\rightarrow}_{j',0}$.

If $\texttt{S}(\alpha^{dispense}_{i',j'}) - \texttt{C}(\alpha^{dispense}_{i,j}) < p^{\rightarrow}_{j,0} + p^{\uparrow} + p^{\rightarrow}_{0,j'}$, then $\texttt{LS}(\alpha^{serve}_{i',j'}) < \texttt{C}(\alpha^{serve}_{i,j})$ and hence due to the *always-equal* constraints on the pipette state (4.3g), must be in the same interval as well as being left aligned. Hence the two serve variables must start at the same time, which filters $\texttt{LS}(\alpha^{serve}_{i',j'}) = \texttt{LS}(\alpha^{serve}_{i,j})$.

If $\texttt{S}(\alpha^{dispense}_{i',j'}) - \texttt{C}(\alpha^{dispense}_{i,j}) \geq p^{\rightarrow}_{j,0} + p^{\uparrow} + p^{\rightarrow}_{0,j'}$ then $\texttt{LS}(\alpha^{serve}_{i',j'}) \geq \texttt{C}(\alpha^{serve}_{i,j})$. Hence it is possible that the two operations are in different intervals of the pipette state and thus the end time cannot yet be further filtered.

□

From Proposition 4.2.2 if there is insufficient time to aspirate between two

dispenses then the serve variables associated with these variables must start together. It is possible to have multiple such pairs in sequence. Let us now consider sequences of such pairs. Let Γ_c be a set of all subsequence in $seq(A)$ such that for any subsequence $\gamma \in \Gamma_c$ all operations in the subsequence require chemical $c \in C$, i.e. $c_{i,j} = c \; \forall (i,j) \in \gamma$ and the cardinality of the subsequence minimally exceeds the capacity of the pipette, i.e., $|\gamma| = L_c + 1$. Also for any sequence γ, let $(i^1, j^1) \in \gamma$ denote the first operation in the sequence, $(i^2, j^2) \in \gamma$ denote the second operation in the sequence and so on.

Proposition 4.2.3. *For every subsequence $\gamma \in \Gamma_c$ of sequence $seq(A)$ for chemical $c \in C$, there must exists at least one consecutive pair of operations $(i,j),(i',j') \in \gamma$ such that $\mathtt{S}(\alpha^{dispense}_{i',j'}) - \mathtt{C}(\alpha^{dispense}_{i,j}) \geq p^{\rightarrow}_{j,0} + p^{\uparrow} + p^{\rightarrow}_{j',0}$*

Proof. For the sake of contradiction assume for a given chemical $c \in C$ that there exists a set of consecutive dispenses $\gamma \in \Gamma_c$ such that for all consecutive operations $(i,j),(i',j') \in \gamma$, the distance between the dispenses is $\mathtt{S}(\alpha^{dispense}_{i',j'}) - \mathtt{C}(\alpha^{dispense}_{i,j}) < p^{\rightarrow}_{j,0} + p^{\uparrow} + p^{\rightarrow}_{j',0}$. From Proposition 4.2.2, we know that the latest start time of all the serve variables associated with all operations in the subsequence must then be at the same time, $\mathtt{LS}(\alpha^{serve}_{i^1,j^1})$. As a result we know that all for all operations in the subsequence the serve variables will at least overlap in the interval $[\mathtt{LS}(\alpha^{serve}_{i^1,j^1}), ..., \mathtt{C}(\alpha^{serve}_{i^1,j^1})]$. However this is a contradiction as the cardinality of the subsequence, $|\gamma| = L_c + 1$, exceeds the carrying capacity for the required chemical, and thus the cumulative expression (4.3h) would be violated. □

Before considering the construction of the solution based on assignment A let us first consider the additional precedence constraints (4.3e) and (4.3f).

Proposition 4.2.4. *Given A, if any of the constraints (4.3e) and (4.3f) are violated then the always equal constraints on the pipette state (4.3g) are also violated.*

Proof. Given assignment A let us assume for the sake of contradiction that there exists an additional precedence constraint that is violated, which does not violate the pipette-state. Let $(i,j),(i+1) \in O$ be the two operations from the violated precedence constraint.

Firstly lets assume that $c_{i,j} = c_{i+1,j}$. Hence the violated constraint is a *start-start* precedence constraint and hence we know that $\mathtt{LS}(\alpha^{serve}_{i+1,j}) < \mathtt{ES}(\alpha^{serve}_{i,j})$. This implies $\mathtt{LS}(\alpha^{serve}_{i+1,j}) < \mathtt{ES}(\alpha^{serve}_{i,j}) < \mathtt{C}(\alpha^{serve}_{i,j}) < \mathtt{C}(\alpha^{serve}_{i+1,j})$. Given the *always-equal* constraints between the pipette state and the serve variables the two serve variables must be in the same interval of the pipette state for the duration of both intervals. However the serve variables are also left aligned and thus must both start at the start of the

interval in the pipette state. As $\texttt{LS}(\alpha^{serve}_{i+1,j}) < \texttt{ES}(\alpha^{serve}_{i,j})$ this violates the *always-equal* constraints.

Thus the only alternative is that $c_{i,j} \neq c_{i+1,j}$. Hence the violated constraint is an *end-start* precedence constraint. As this is violated we know that $\texttt{ES}(\alpha^{serve}_{i+1}) < \texttt{C}(\alpha^{serve}_{i,j})$. Hence $\texttt{LS}(\alpha^{serve}_{i+1,j}) < \texttt{C}(\alpha^{serve}_{i,j}) < \texttt{C}(\alpha^{serve}_{i+1,j})$. Thus there is an interval of time where the serve variables must overlap. This violates the *always-equal* constraints on the serve variables, as the pipette state must always equal the value associated with their respective chemicals for the duration of the serve variables. This is a contradiction. □

Using all of the previous propositions it is now possible to prove the following theorem.

Theorem 4.2.5. *Given assignment A a solution can be constructed by fixing the start time of all serve variables to their latest start time after propagation.*

Proof. From Proposition 4.2.4 we do not need to consider if the additional precedence constraints are violated as so too will the constraints on the pipette state. From Proposition 4.2.2 we know that for any assignment of the start time of the serve variables less than its latest starting time after propagation will satisfy the conjunction of the *end-at-end* constraints and the minimum length on the serve variables. Thus we only need to consider the always-equal constraints and the cumulative constraint.

From Proposition 4.2.2, we know that for consecutive operations if $c_{i,j} = c_{i',j'} \wedge \texttt{S}(\alpha^{dispense}_{i',j'}) - \texttt{C}(\alpha^{dispense}_{i,j}) < p^{\rightarrow}_{j,0} + p^{\uparrow} + p^{\rightarrow}_{j',0}$ then $\texttt{LS}(\alpha^{serve}_{i',j'}) = \texttt{LS}(\alpha^{serve}_{i,j})$, otherwise $\texttt{LS}(\alpha^{serve}_{i',j'}) = \texttt{S}(\alpha^{dispense}_{i',j'}) - p^{\uparrow} - p^{\rightarrow}_{0,j'}$. Let us now fix $\texttt{S}(\alpha^{serve}_{i,j}) = \texttt{LS}(\alpha^{serve}_{i,j})$ for all $(i,j) \in O$. This also fixes the intervals of the pipette state. For all consecutive operations $(i,j),(i',j') \in O$ where $\texttt{S}(\alpha^{dispense}_{i',j'}) - \texttt{C}(\alpha^{dispense}_{i,j}) \geq p^{\rightarrow}_{j,0} + p^{\uparrow} + p^{\rightarrow}_{j',0}$, we know that $\texttt{S}(\alpha^{serve}_{i',j'}) \geq \texttt{C}(\alpha^{serve}_{i,j})$ and hence are in different intervals of the pipette state. On the other hand if $c_{i,j} = c_{i',j'} \wedge \texttt{S}(\alpha^{dispense}_{i',j'}) - \texttt{C}(\alpha^{dispense}_{i,j}) < p^{\rightarrow}_{j,0} + p^{\uparrow} + p^{\rightarrow}_{j',0}$ then $\texttt{S}(\alpha^{serve}_{i',j'}) = \texttt{S}(\alpha^{serve}_{i,j})$, which satisfies the constraints on the pipette state as the two serve variables are left aligned and have the same value.

Finally from Proposition 4.2.3, we know that given the $seq(A)$ there does not exist a subsequence $\gamma \in \Gamma_c$, where all consecutive pairs of operations $(i,j),(i',j') \in \gamma$ have dispenses that are separated by $\texttt{S}(\alpha^{dispense}_{i',j'}) - \texttt{C}(\alpha^{dispense}_{i,j}) < p^{\rightarrow}_{j,0} + p^{\uparrow} + p^{\rightarrow}_{j',0}$. Hence we know that there is at most L_c serve variables in each interval of the pipette state of value $c \in C$. Therefore the cumulative constraint is satisfied and

thus the construction satisfies all of the constraints and is thus a solution. □

Theorem 4.2.5 proves that it is possible to schedule all of the dispense variables first and then fix the serve variables afterwards. Thus it is possible to separate the search into two search phases. The first phase searches only on the dispense variables. The second search phase simply assigns all the remaining variables to their latest starting time.

An alternative construction that does not guarantee a solution is based on assigning the serve variables to their earliest start times according to the SetTimes strategy (recapped in Godard et al. (2005)). This construction is important as SetTimes is used extensively by CP Optimizer. These strategies consider interval variables by increasing the indicative start (or end) times and trying to schedule them as close as possible to their indicative times. For objectives such as makespan this means that variables are assigned to their earliest start time. As described in Laborie et al. (2018), when a failure occurs, in the right branch the decision variable is marked "unselectable" and will remain so until constraint propagation removes from the current domain of the variable the literal that was used on the left branch.

Proposition 4.2.6. *Given A, using SetTimes to fix the start times of serve variables is incomplete.*

Proof. The proof is a simple counter example that is visualised in Figure 4.6. Consider a problem with three jobs $|J| = 3$, each job having one operation each $|O_j| = 1$ for all $j \in J$. Let $p^{\uparrow} = 20$, $p^{\downarrow} = 10$ and $p^{\rightarrow}_{j,j'} = |j - j'|$ for all $j, j' \in J \cup \{0\}$. All jobs require the same chemical c, which has a limit $L_c = 2$. Fix the start time of the dispense intervals for operations $(1,1), (1,2), (1,3)$ to 21, 61, 73 respectively. From the *end-at-end* constraints, the end of the serve variables are fixed. Moreover after propagation $S(\alpha^{serve}_{1,1}) = 0$. As $S(\alpha^{dispense}_{1,3}) - C(\alpha^{dispense}_{1,2}) < p^{\rightarrow}_{2,0} + p^{\uparrow} + p^{\rightarrow}_{0,3}$, we know that due to the *always-equal* constraints on the state function that $LS(\alpha^{serve}_{1,2}) = LS(\alpha^{serve}_{1,3})$ and $ES(\alpha^{serve}_{1,2}) = ES(\alpha^{serve}_{1,3})$. Note that an assignment of $ES(\alpha^{serve}_{1,2}) = 0$ would clearly violate the cumulative constraint as this would propagate $ES(\alpha^{serve}_{1,3}) = 0$ and thus the three serve variables would overlap and exceed the carrying limit of their required chemical. Recall that in CP constraints are considered separately and domain reductions that rely on their conjunction are not inferred. Hence despite these failures the domains on the early start of the serve variables would not be reduced as they rely on the conjunction of the cumulative constraints and the *always-equal* constraints on the pipette state. Therefore $\alpha^{serve}_{1,2}$ and $\alpha^{serve}_{1,3}$ both have the same earliest and latest starting times and assigning either variable to their ear-

liest starting time will lead to a failure, SetTimes will fail. Furthermore as a solution exists where $ES(\alpha_{1,2}^{serve}) = ES(\alpha_{1,3}^{serve}) = 32$, SetTimes is incomplete. □

CP Optimizer has a time-enumerated version of the SetTimes strategy. Hence their default search is still complete. In the time-enumerated version when a failure occurs, in the right branch the literal assigned on the left branch is simply removed from the domain. For example, for the counter example used in the proof of Proposition 4.2.6, after the solver tries and fails to assign both $\alpha_{1,2}^{serve}$ and $\alpha_{1,3}^{serve}$ to start at time 0, the solver will then try to assign one of these variables to time 1, and then time 2 and so on until a feasible solution is found. From Theorem 4.2.5 we know that a feasible construction is possible for assignment A when the serve variables are assigned to their latest starting time so clearly the time-enumerated version of the SetTimes strategy will also lead to a feasible construction eventually.

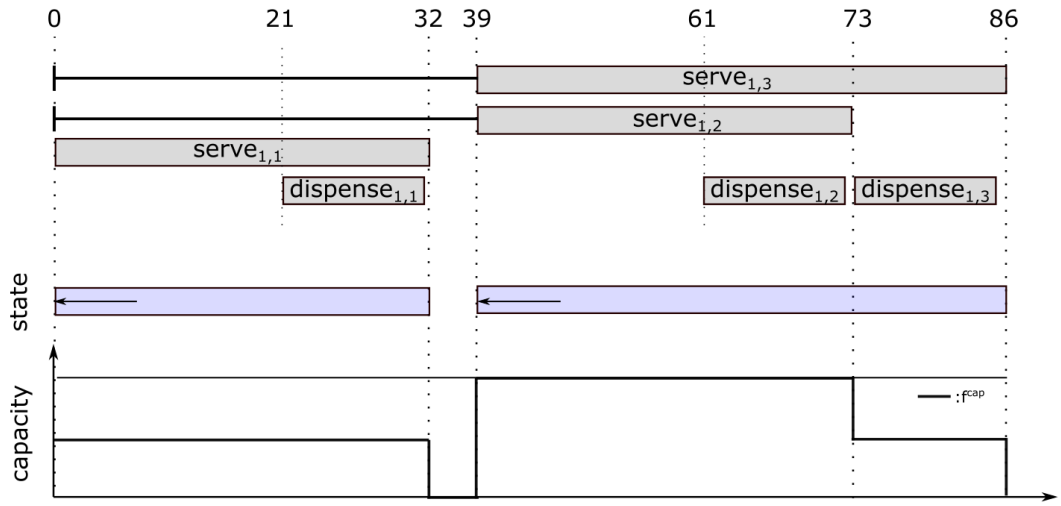


Figure 4.6: Counter example in proof of Proposition 4.2.6

4.3 Lower Bounds

Here we consider how lower bounds can be obtained to the problem based on the (1) the temporal network, and (2) the minimum resource required on the robot.

4.3.1 Temporal-Based

The first lower bound is based on the precedences, or the temporal network, of the problem. Intuitively the temporal-based lower bound represents the least amount of time required to schedule a single job j from the set of jobs J. Clearly it is not

possible to schedule all jobs in less time than it is to schedule a single job in that set. Firstly we define a strengthened minimum time lag, $\overline{\ell}_{i,j}^{min}$ between consecutive operations, (i,j) and $(i+1,j)$ from the same job, $j \in J \setminus \{N_j\}$,

$$\overline{\ell}_{i,j}^{min} := \begin{cases} \max(\ell_{i,j}^{min}, p^{\downarrow} + p^{\uparrow} + p_{0,j}^{\rightarrow} + p_{j,0}^{\rightarrow}) & \text{if } c_{i,j} \neq c_{i+1,j} \\ \max(\ell_{i,j}^{min}, p^{\downarrow}) & \text{otherwise} \end{cases}$$

If consecutive operations from the same job require different chemicals then the robot must travel back to the vials to aspirate more chemical between these two operations. On the other hand, if the consecutive operations from the same job require the same chemical, then the robot still must dispense the first unit of chemical before it can dispense the second. Clearly $\overline{\ell}_{i,j}^{min} \geq \ell_{i,j}^{min}$ for all $i \in O_j$ and $j \in J$. Now we can express the temporal-based lower bound as follows,

$$LB_{temp} = \max_{j \in J}(p^{\uparrow} + p_{0,j}^{\rightarrow} + \sum_{i \in O_j \setminus \{n_j\}} \overline{\ell}_{i,j}^{min} + p^{\downarrow}) \tag{4.5}$$

4.3.2 Resource-Based

The second lower bound is based on the minimum time the robot requires to complete all the possible operations. For this lower bound we ignore the precedences, and thus the minimum and maximum time lags, between consecutive operations from the same job. Hence it is now possible to schedule operation $i+1$ before operation i. We first define the minimum number of dispenses and aspirates of chemical $c \in C$ as $n_c^{\downarrow} := |O^c|$ and $n_c^{\uparrow} := \lceil \frac{|O^c|}{L_c} \rceil$ respectively. The resource-based lower bound is then defined as follows,

$$LB_{res} = \sum_{c \in C} (\underbrace{n_c^{\downarrow} * p^{\downarrow}}_{\text{(i)}} + \underbrace{n_c^{\uparrow} * p^{\uparrow}}_{(ii)} + \underbrace{\sum_{k \in [1, n^{\uparrow}]} \overset{(k-1)L_c}{\max_{(i,j) \in O^c}} (p_{0,j}^{\rightarrow} + p_{j,0}^{\rightarrow})}_{(iii)}) - \underbrace{\max_{j \in J} p_{j,0}^{\rightarrow}}_{(iv)} \tag{4.6}$$

where the operator $\overset{k}{\max}$ refers to the $k'th$ largest element in the specified set. The first two terms, (i) and (ii), represent the minimum amount of time required dispensing and aspirating, respectively. The third term, (iii), represents a valid lower bound on the amount of travel time required. For each chemical, the operations are effectively placed in groups of size L_c (with one group representing the

remainder) according to how much time it takes to travel to and from the job. For each of these groups we only consider the maximum time. The main idea, is that we might travel past the other locations of operations in this group during this travel time. This is true when the jobs are arranged in a straight line as depicted in Figure 4.1. The forth term, (iv), subtracts the time to return from the furthest job to the vial to account for the fact that the makespan is measured from the end of the last dispense operation.

4.4 Computational Study

4.4.1 Data

To test the different models we have generated a data set of 24 instances where parameters are chosen from intervals that try to replicate the structure of the real-world problem while also incrementally increasing the number of jobs and operations being considered.

All jobs in an instance have the same number of operations denoted by m, i.e., $|O_j| = m$ for all $j \in J$. The *size* of an instance is represented by a tuple, $(n, m, |C|)$, which describes the number of jobs, number of operations per job and the number of unique chemicals respectively. For all instances the processing times are as follows, $p^{\rightarrow}_{j,j'} = |j' - j|$ for all $j, j' \in J \cup \{0\}$, $p^{\downarrow} = 10$, $p^{\uparrow} = 20$. The carry limit L_c for individual chemicals $c \in C$ within an instance is taken as a random integer between 2 and 5. The minimum time lags, $\ell^{min}_{i,j}$ are selected at random from the interval $[p^{\downarrow}, \lambda * (p^{\downarrow} + p^{\uparrow} + 2n)]$, to allow the minimum time lags to increase as the number of jobs increases, where λ is a scaling parameter. In Edwards et al. (2018) the 18 instances considered all had $\lambda = n$. This resulted in instances where the temporal lower bound dominated the resource lower bound. In this chapter we consider six more instances where the resource lower bound dominates the temporal lower bound and for these instances $\lambda = n/4$.

In practice it is trivial to design the parameters of the instance to ensure feasibility. As such, in this study we restrict ourselves to feasible instances. Consequently, the minimum time lag is used when constructing the corresponding maximum time lag, by introducing factor $\epsilon_{i,j} = max(\ell^{min}_{i,j}, p^{\downarrow} + 2n + p^{\uparrow})$, which is the maximum of the minimum time lag and the minimum amount of time the robot requires to complete consecutive operations for the furthest job if the operations require different chemicals. The maximum time lags, $\ell^{max}_{i,j}$ and are selected at random be-

tween $\ell_{i,j}^{max} \in [\epsilon_{i,j}, n\epsilon_{i,j}]$. This allows the maximum time lags to increase when more jobs are considered. We round the minimum time lags and maximum time lags.

4.4.2 Preprocessing

As noted in the precedence based lower bound section, Section 4.3.1, it is possible to strengthen the minimum time lags between consecutive operations from the same job. Before all of the approaches mentioned we complete a preprocessing step where the minimum-time lags are replaced by the strengthened minimum time lags. All approaches benefit from this preprocessing step however no formal study was completed to demonstrate the extent of this improvement.

4.4.3 Experimental Set Up

Experiments have been tested under the Windows 10 (64-bit) operating system with 8GB RAM and Intel® Core™ i7-3537U, 2.5GHz processor. The MIP model was implemented in Gurobi 7.0.2 and the CP model was implemented in IBM ILOG CPLEX Optimization Studio V12.8.0 CP Optimizer. The MIP and CP approaches for the full problems are given 4 threads and 10 minutes wall-time.

Table 4.1 summarises the different approaches being tested. Recall that for CP3 we proved that it is possible to search first on the dispense variables and then once they are all fixed search on the serve variables. CP3X is the CP3 model where a search phase is added to complete the default search on the dispense variables. Hence CP3X will use the time-enumerated version of SetTimes to schedule the serve variables once the dispense variables have been fixed. CP3Y is the CP3 model where again a search phase is added to complete the default search on the dispense variables, after which a second phase is used to assign the serve variables to their latest starting time. As of version 12.8.0 of CP Optimizer it is not possible to directly implement this type of search phase on interval variables. Hence for each serve variable an integer variable is introduced and a constraint is used to force the start of the serve variable to equal this integer variable. It is then possible to add a search phase on these integer variables and assign each to the largest value in their domain.

4.4.4 Results

Unless specified otherwise, a result is displayed in bold if it corresponds to the best solution found and an asterisk (*) is used to indicate that corresponding approach

Table 4.1: A summary of the different approaches being tested.

Problem	Acronym	Section	Notes
Full	MIP	B.1	
	CP1	4.2.1	
	CP2	4.2.2	
	CP3	4.2.3	
	CP3X	4.2.3	Default search to schedule serve variables
	CP3Y	4.2.3	Serve variables assigned to their LST

could prove that the solution found is the optimal solution within the specified time-limit.

The results of the complete approaches are summarised in Table 4.2. The column lb represents the best known lower bound on the instance. For all models ub and $t(s)$ represent the best objective value found and the wall time that the algorithm requires to prove optimality respectively. The upper bound column is left empty if no feasible solution is found. The time column is left empty if the algorithm does not prove optimality and thus terminates after 10 minutes. Finally the MIP model has the additional column, $lazy$, which represents the number of lazy-constraints that were added in the MIP approach.

Table 4.2: Best Solution (10 minutes)

Inst.	Size	lb	MIP			CP1		CP2		CP3		CP3X		CP3Y	
			lazy	ub	t(s)	ub	t(s)	ub	t(s)	ub	t(s)	ub	t(s)	ub	t(s)
1	(3-5-2)	361	7	**361***	3.69	**361**	-	**361***	2.51	**361***	0.37	**361***	0.57	**361***	0.74
2	(3-5-3)	477	-	**477***	3.12	**477**	-	**477***	4.89	**477***	0.93	**477***	0.75	**477***	1.02
3	(3-5-5)	588	-	**588***	8.62	**588**	-	**588***	9.82	**588***	1.43	**588***	1.51	**588***	2.43
4	(5-5-2)	707	54	716	-	716	-	**707***	10.52	**707***	1.71	**707***	1.15	**707***	2.43
5	(5-5-3)	696	12	749	-	707	-	**696***	28.00	**696***	4.05	**696***	3.25	**696***	4.20
6	(5-5-5)	644	9	649	-	644	-	**644***	11.52	**644***	3.47	**644***	2.68	**644***	3.70
7	(5-10-2)	1187	-	-	-	1341	-	1359	-	**1327**	-	1339	-	**1327**	-
8	(5-10-3)	1261	-	-	-	1267	-	1267	-	**1261***	463.00	**1261**	-	1273	-
9	(5-10-5)	1335	-	-	-	1475	-	**1335**	-	**1335***	187.16	**1335***	159.44	**1335***	302.07
10	(10-10-2)	2891	-	-	-	3077	-	**2891***	324.82	**2891***	33.60	**2891***	53.02	**2891***	186.03
11	(10-10-3)	2700	-	-	-	3076	-	2819	-	2767	-	**2756**	-	2763	-
12	(10-10-5)	2973	-	-	-	3346	-	3024	-	2980	-	2978	-	**2974**	-
13	(20-10-2)	4834	-	-	-	6608	-	5491	-	**5200**	-	5233	-	5240	-
14	(20-10-3)	3706	-	-	-	6792	-	4405	-	4265	-	**4226**	-	4307	-
15	(20-10-5)	4468	-	-	-	7288	-	5206	-	5173	-	5239	-	**5130**	-
16	(15-20-2)	10349	-	-	-	11404	-	**10349***	591.46	**10349***	106.94	**10349***	142.28	**10349***	32.72
17	(15-20-3)	10707	-	-	-	13836	-	**10707***	542.80	**10707***	45.28	**10707***	19.12	**10707***	9.38
18	(15-20-5)	11234	-	-	-	17081	-	11571	-	**11234***	196.76	**11234***	12.82	**11234***	22.22
19	(30-20-2)	15036	-	-	-	-	-	34522	-	17890	-	17265	-	**17197**	-
20	(30-20-3)	15474	-	-	-	46004	-	-	-	19100	-	18070	-	**17970**	-
21	(30-20-5)	17984	-	-	-	42352	-	40764	-	22447	-	21343	-	**21125**	-
22	(20-30-2)	23343	-	-	-	-	-	25277	-	**23343***	487.49	23353	-	**23343***	69.07
23	(20-30-3)	25087	-	-	-	-	-	90150	-	**25087***	350.66	**25087***	154.99	**25087***	71.80
24	(20-30-5)	23048	-	-	-	-	-	197569	-	24123	-	23452	-	**23344**	-

The lower bounds determined by the different methods are summarised in Table 4.3. The column *temp-lb* and *res-lb* are the temporal and resource-based lower bounds respectively. The remaining six columns represent the lower bounds obtained by the different approaches after 10 minutes solve time respectively. These values are left blank if the lower bounds are the same as the temporal based lower bound.

Table 4.3: Lower Bounds

Inst.	Size	temp-lb	res-lb	MIP	CP1	CP2	CP3	CP3X	CP3Y
1	(3-5-2)	292	270	**361**	301	**361**	**361**	**361**	**361**
2	(3-5-3)	366	250	**477**	-	**477**	**477**	**477**	**477**
3	(3-5-5)	396	342	**588**	-	**588**	**588**	**588**	**588**
4	(5-5-2)	692	514	-	-	**707**	**707**	**707**	**707**
5	(5-5-3)	664	490	-	-	**696**	**696**	**696**	**696**
6	(5-5-5)	624	538	-	-	**644**	**644**	**644**	**644**
7	(5-10-2)	1134	1156	-	1144	1164	1177	**1187**	1170
8	(5-10-3)	1045	1022	-	-	1073	**1261**	1111	1076
9	(5-10-5)	1178	1178	-	-	1193	**1335**	**1335**	**1335**
10	(10-10-2)	2833	1788	-	-	**2891**	**2891**	**2891**	**2891**
11	(10-10-3)	2664	2084	-	-	2698	**2704**	**2704**	**2704**
12	(10-10-5)	**2973**	2188	-	-	-	-	-	-
13	(20-10-2)	2130	**4834**	4796	2136	2159	3461	3461	3461
14	(20-10-3)	2229	**3706**	3487	-	2269	2279	2279	2279
15	(20-10-5)	2025	**4468**	4307	2046	2069	2920	2920	2920
16	(15-20-2)	**10349**	6216	-	-	-	-	-	-
17	(15-20-3)	**10707**	6150	-	-	-	-	-	-
18	(15-20-5)	**11234**	6672	-	-	-	-	-	-
19	(30-20-2)	7327	**15036**	13589	-	7364	10693	10693	10693
20	(30-20-3)	7494	**15474**	14139	7496	-	11186	11186	11186
21	(30-20-5)	7641	**17984**	17824	7647	-	14195	14195	14195
22	(20-30-2)	23324	10974	-	-	23336	**23343**	**23343**	**23343**
23	(20-30-3)	**25087**	11798	-	-	-	-	-	-
24	(20-30-5)	**23048**	119176	-	-	-	-	-	-

Table 4.4 compares the solutions found by the CP models after 1 minute solve time. The column *lb* refers to the best lower bound found by any model on the full problem.

In Table 4.2 one can clearly see gradual improvement on the results from the MIP model to CP3Y, i.e., from left to right. The MIP model does not scale well. The number of arc variables is in the order of the square of the number of operations. Furthermore the big M notation in the linking constraint is very weak. We believe that these two factors limit the viability of this approach.

Although the heuristics in CP Optimizer allow the CP1 model to find a number of feasible solution, CP1 also has a problem with scaling. For instances 19 and 22-

Table 4.4: Comparison of CP approaches (1 minute)

Inst.	Size	lb	CP1	CP2	CP3	CP3X	CP3Y
1	(3-5-2)	361	**361**	**361***	**361***	**361***	**361***
2	(3-5-3)	477	**477**	**477***	**477***	**477***	**477***
3	(3-5-5)	588	**588**	**588***	**588***	**588***	**588***
4	(5-5-2)	707	716	**707***	**707***	**707***	**707***
5	(5-5-3)	696	707	**696***	**696***	**696***	**696***
6	(5-5-5)	644	**644**	**644***	**644***	**644***	**644***
7	(5-10-2)	1167	1396	1392	1351	1339	**1327**
8	(5-10-3)	1073	1372	1296	1296	**1282**	**1282**
9	(5-10-5)	1211	1513	1378	1371	**1360**	**1360**
10	(10-10-2)	2891	3283	2896	**2891***	**2891***	2912
11	(10-10-3)	2700	3158	2861	2816	2833	**2769**
12	(10-10-5)	2973	3599	3068	3059	**2978**	2980
13	(20-10-2)	4386	10245	9371	5430	**5349**	5355
14	(20-10-3)	3706	8197	4643	4584	**4428**	4462
15	(20-10-5)	4468	9416	5472	5431	5390	**5247**
16	(15-20-2)	10349	15412	34930	10509	10473	**10349***
17	(15-20-3)	10707	21173	49024	**10707***	**10707***	**10707***
18	(15-20-5)	11234	22649	92468	11641	**11234***	**11234***
19	(30-20-2)	15036	-	-	32982	19361	**19110**
20	(30-20-3)	15474	-	-	20277	20138	**19863**
21	(30-20-5)	17984	-	-	35517	23571	**23480**
22	(20-30-2)	23343	-	96632	24176	23458	**23343***
23	(20-30-3)	25087	-	-	26615	25855	**25097**
24	(20-30-5)	23048	-	-	44194	24013	**23816**

24 CP1 is unable to even find a feasible solution within 10 minutes. Furthermore CP1 is not able to prove any instances to optimality and is only able to marginally improve the lower bounds from the *temp-lb* in 6 instances. This highlights the weak propagation between the two sets of variables, (1) the aspirate and carry variables and (2) the cover and dispense variables.

CP2 significantly improves the results from CP1. CP2 is able to prove optimality for 9 instances. It also either finds the same or better solution of both MIP and CP1 on all instances except instance 20 where it cannot find a feasible solution within 10 minutes. CP2 also manages to improve the lower bound from the temporal lower bound on 16 of the instances. Clearly CP2 appears to considerably outperform CP1, which is attributed in the stronger propagation between aspirate and dispense variables, as well as using a single cumulative expression to keep track of the chemical being carried.

CP3 considerably improves the results from CP2. In fact, CP2 does not out perform CP3 on a single instance. For the larger instances the improvement obtained by CP3 is very significant, for example consider the improvement for in-

stance 24 from 197,569 to 24,123. Furthermore CP3 manages to prove 14 instances to optimality, five more than CP2. The improvement between CP3 and CP2 can be attributed to the stronger propagation achieved through the additional precedence constraints that provide more information to the precedence graph, the normalised cumulative expression being able to use the edge finding propagator, and increasing the size of the serve variables. Remodelling the serve variable had the additional benefit of making the aspirate variable and the position state redundant and hence reduced the size of the model. The stronger propagation is reflected by the tighter lower bounds found by CP3 compared to CP2. This is particularly clear on the instances where the resource lower bound dominates the temporal lower bounds (instances 13-15, 20-22).

The search phases in general allow for better solutions to be found faster. This is true for both the solutions found after 1 and 10 minutes but is particularly clear on larger instances of the former. For instances 15-21,23-24 in Table 4.4 the solution found by CP3 is not better than the solution found by CP3X, which is itself not better than the solution found by CP3Y. Despite having to introduce additional integer variables to implement CP3Y, in general it tends to find the best solutions. Intuitively this makes sense as once the dispense variables are fixed, a solution is constructed directly by assigning the start time of the serve variables to their latest starting time. Whereas we know that there are situations for which CP3X must enumerate the time horizon to obtain a solution after the primary search phase, which would logically take more time. The integer variables introduced by CP3Y do incur some additional overhead. As search phases are ignored during FDS, as of CP Optimizer version 12.8, these additional variables result in a larger search tree.[1] This is highlighted by the inferior lower bounds found by CP3Y on instances 7 and 8 compared to CP3 and CP3X. Furthermore CP3X and CP3 both obtain a better solution on instance 14 than CP3Y. We suspect this is also a result of the fact that the CP Optimizer uses an adaptive LNS during the search, so accepting an incumbent solution may lead the search to different areas. It is believed that the results of CP3Y could be even further improved if CP Optimizer allowed for the desired search phase to be implemented directly on the serve variables without the need to introduce additional integer variables.

[1] Thank you to Dr Philippe Laborie for making this clarification.

4.5 Conclusion

In this chapter we introduced the LHRSP, which is a sub-problem of the real-world problem. We proposed and studied three different CP Optimizer models (CP1, CP2, CP3). A major contribution of this chapter is that for CP3, we proved that it is possible to only search over a subset of the variables and then extend this partial assignment to a complete solution. We compared two different methods for implementing the corresponding seach phases (CP3X and CP3Y).

The best CP model, CP3, demonstrates the benefits that can be obtained by modelling problems with a strong temporal network of which the maximally connected components represent logical choices for schedule generation schemes. Furthermore CP3 demonstrates the benefits that can be obtained through modelling cumulative expressions such that the edge-finder constraint propagator can be used. Finally CP3X and CP3Y also demonstrate how search phases can be used to only search on a subset of decision variables. This is particularly true for CP3Y, which constructs a solution directly once the first search phase is completed. This resulted in a considerable reduction of the time taken to find good quality solutions.

In the next chapter we generalise the insight used in this chapter that once a subset of the variables are fixed and constraint propagation reaches a fixed point, it is always possible to extend the partial assignment to a complete solution. We then consider a more general version of the LHRSP, and demonstrate that searching over a subset of the variables is key to good solver performance.

CHAPTER 5

Driving Variables and Lower-Upper Consistency

5.1 Introduction

In many practical applications of constraint programming (CP) it is often necessary to introduce auxiliary variables into a model either because it is difficult to express the constraints at all in terms of the existing variables, or to allow the constraints to be expressed in a form that would propagate more efficiently (Smith, 2006). As seen in the previous chapter, an immediate consequence of increasing the number of variables however is that unless the additional variables are dealt with effectively, they can dramatically increase the size of the search tree as there are more variables to choose from when branching. As a result, there has been much research in the literature that explores how assignments over subsets of variables can be extended to complete solutions.

In this chapter we formalise the notion of *driving variables*. To our knowledge, the term driving variables was first used by Wallace (1996) to informally describe the set of variables whose "final values define a complete solution". To that effect, we consider a subset of the variables to be driving variables if once they are assigned values and constraint propagation reaches a fixed point, a search strategy is known to always generate a complete assignment without backtracking. Note that

there may be more than one subset that can be considered as the driving variables. A key distinguishing characteristic of driving variables is that once the driving variables have been assigned, it is possible to reason about the domains of the remaining unfixed variables at a fixed point due to the local consistency rules of the various filtering algorithms independent of the constraint structure. Although the chapter introduces and distinguishes driving variables from related concepts in detail, a very high-level summary is as follows.

- Functional dependencies, a concept more associated with relational databases (Watt and Eng, 2014), rely on the filtering of individual constraints to fix dependent variables independent of the structure of the constraints. However they result in all variables becoming fixed through propagation, which is often too strict of a condition to be applied in many situations.
- W-cutsets of constraint graphs (Dechter, 2006), and their generalisations, use relational consistency rules to reason about complete assignments, yet this reasoning is restrictively based on the structure of the corresponding constraint graph. Consequently, this excludes other local consistency rules such as the one introduced in this chapter.
- Backdoors to typical case complexity (Williams et al., 2003) is a general framework for reasoning about problems where for any assignment over a subset of the variables there exists a polynomial-time algorithm that can either extend the assignment to a full solution or prove that a solution does not exist. When considering backdoors with respect to CP, a significant limitation arises when considering the fact that in general reaching a fixed point by constraint propagation requires pseudo-polynomial-time, which is technically not considered polynomial (Hall, 2013). Consequently, by definition, backdoors exclude the use of constraint propagation when extending the partial assignment to a full solution. This is a significant restriction. If constraint propagation does not reach a fixed point, then it is not possible to reason about the status of the domains of the remaining variables based on the local consistency rules of the filtering algorithms of the individual constraints.

To demonstrate how it is possible to reason about complete solutions based on local consistency rules independent of constraint structure, we introduce the novel concept of *Lower-Upper (LU) consistency*. LU consistency is a local consistency rule requiring that for a given partition of variables in the scope of a constraint, it is always possible to satisfy the constraint by assigning the first set of variables to their lower-bound and the second set of variables to their upper-bound. If all con-

straints are LU-consistent for the same partition of variables, a complete solution can then be determined at a fixed point through the correct assignment of variables in these sets. Admittedly, LU consistency is a very strict level of local-consistency and most known filtering algorithms cannot ensure that level of consistency in general. However, as will be demonstrated in this chapter, there are many useful scenarios in which special cases of known filtering algorithms and consistency-checks can become LU-consistent. In this context, driving variables can be seen as the set of variables that once set, ensure that all constraints become LU-consistent for a unique partition of the remaining variables.

In many applications of CP there exist sufficiently tight bounds on the size of domains, in which case one could reasonably argue that fixed point computation is tractable. In such a case, as proven in this chapter, driving variables are then equivalent to (strong) backdoors. However, one of the strengths of CP comes from its ability to efficiently reason over variables with very large domains. Such examples frequently arise in the field of scheduling, which is one of the most successful application areas of CP (Laborie et al., 2018). In this chapter we motivate why constraint-based scheduling approaches can often have very large domain sizes, which serves to highlight the importance of considering fixed point complexity explicitly and thus the need to differentiate driving variables from backdoors. We then prove for many important global constraints in scheduling that there are numerous special cases where various partitions of variables are LU-consistent. The global constraints considered include four of the major classes of constraint types in CP Optimizer: precedences constraints, cumulative function expressions, state functions, and non-overlapping sequences.

The remainder of the chapter is structured as follows. Section 5.2 introduces necessary definitions and terminology to allow a formal definition of driving variables and LU consistency to be stated. Section 5.3 reviews the related concepts from the literature and clearly motivates and distinguishes the concepts introduced in this chapter. Section 5.4 proves that a number of existing filtering algorithms and consistency-checks from the scheduling-literature are LU-consistent in various special-cases and for specific partitions. Section 5.5 then demonstrates the practical benefit that can be obtained by correctly accounting for driving variables on the real-world scheduling problem that motivated this work. Finally, Section 5.6 provides a conclusion and suggests directions for future work.

5.2 Definitions

This section is divided into three parts. Firstly, we introduce some notation and general definitions of constraint satisfaction problems (CSPs). Secondly, we define the novel local consistency measure, LU-consistency, and demonstrate how LU consistency can be used to extend partial assignments to complete solutions. Thirdly, we formally define the term driving variables, and provide the main theorem of the chapter which relates driving variables and LU-consistency.

5.2.1 Constraint Satisfaction Problems

A CSP is a tuple $\mathcal{P} := (\mathcal{V}, \mathcal{C}, \mathcal{D})$, where $\mathcal{V}$ denotes the set of decision variables, $\mathcal{D}$ the set of domains, and $\mathcal{C}$ the set of constraints. Each variable $x \in \mathcal{V}$ is associated with a finite domain, $dom(x) \in \mathcal{D}$, of potential values. Each constraint $C \in \mathcal{C}$ is a constraint over some subset of variables, denoted $vars(C) \subseteq \mathcal{V}$, referred to as the *scope* of the constraint. A constraint specifies the allowed combination of values for variables in this scope. Given a constraint C, we use the notation $t \in C$ to denote a tuple t, which is an assignment of a value to each of the variables in $vars(C)$ that satisfies the constraint C. We use the notation $t[x]$ to denote the value assigned to variable $x \in vars(C)$ by the tuple $t \in C$. A *value assignment* $A : X \to \mathcal{D}$ is a mapping defined on a subset of variables, $X \subseteq \mathcal{V}$, to values in the appropriate domains. An assignment *satisfies* constraint C if $vars(C) \subseteq X$ and there exists $t \in C$ such that $t[x] = A[x]$ for all $x \in vars(C)$. The objective is to find an assignment over all of the variables that satisfies all of the constraints.

We assume the domains of variables are totally ordered. The minimum and maximum values in the domain $dom(x)$ of a variable x are denoted by $\min(dom(x))$ and $\max(dom(x))$, and the interval notation $[a, b)$ is used as a shorthand for the set of values $\{a, a+1, ..., b-1\}$.

CP is a framework for solving CSPs by interleaving *backtracking tree search* and *constraint propagation*. Backtracking tree search works by iteratively decomposing the problem into possible subproblems. In this chapter it is sufficient to consider sub-domain subproblems, where problem $\mathcal{P}_1$ is a subproblem of $\mathcal{P}_2$, denoted $\mathcal{P}_1 \subseteq \mathcal{P}_2$, if both problems have identical variable and constraint sets, and where each value in the domain of variables in P_1 is also in the domain of the associated variable in P_2. Constraint propagation is the process of eliminating from the domains of the variables those values, referred to as *inconsistent values*, that can never participate in a solution. Algorithms that determine which values are

inconsistent with respect to constraints are known as *filtering algorithms* for the constraint. Filtering algorithms are designed to be fast, i.e. have polynomial-time complexity and in many practical cases typically linear or log-linear, as they are called frequently throughout search. Consequently, it is often too expensive for filtering algorithms to remove all inconsistent values, i.e., they are incomplete. Instead filtering algorithms remove inconsistent values according to certain *local consistency rules*. The filtering algorithms for the different constraints iterate until either none of the filtering algorithms can prune a domain according to their local consistency rules (a state that is known as a *fixed point*) or at least one of the domains of the variables become empty (a state that is known as a *failure*). To make explicit this process of constraint propagation, if $\mathcal{P}$ is the current problem being solved, then we denote the problem at fixed point as $\overline{\mathcal{P}}$ and observe that $\overline{\mathcal{P}} \subseteq \mathcal{P}$.

At a fixed point the search branches by assigning an unfixed variable a value from its domain according to some predescribed rule, known as a *search strategy*. More formally, a search strategy is a tuple of two functions. The first function defines a *variable selection strategy*, $f^{variable}$, which takes the set of unfixed variables as input and return a single variable from that set. The second function defines a *value selection strategy*, f^{value}, which returns a value from the domain of the selected variable. At a fixed point, search branches by assigning the selected variable x to the selected value $v \in dom(x)$ based on the search strategy, and then again completing constraint propagation. Let $\mathcal{P}_{|x=d} \subseteq \mathcal{P}$ be the subproblem created by assigning $x = d$. Similarly, let $\mathcal{P}_{|A}$ be the subproblem created by assignment $A : X \to D$ over a subset of variables $X \subseteq \mathcal{V}$. If propagation fails, then the search backtracks. This process is continued until a complete assignment is found or the entire tree is exhausted.

For optimisation problems, once a solution is found an extra constraint is added to the model to ensure only solutions with better objectives than the incumbent can be found. Once the search tree is exhausted the last solution to be found is guaranteed to be an optimal solution. Hence unless otherwise stated in this chapter we will simply consider satisfaction versions of the problem.

5.2.2 LU Consistency

First we define the novel local consistency measure, LU consistency, and demonstrate how LU consistency can be used to reason about complete solutions.

Definition 5.2.1 (LU consistency)**.** A constraint $C \in \mathcal{C}$ is said to be *LU-consistent* for the given partition of variables $L, U \subseteq vars(C)$ if there exists $t \in C$ such that

$t[x] = \min(dom(x))$ for all $x \in L$ and $t[y] = \max(dom(y))$ for all $y \in U$.

If all constraints are LU consistent for the same disjoint partition of the variables, then at a fixed point a complete assignment can be obtained as follows,

Proposition 5.2.2. *If all constraints of problem $\mathcal{P}$ are LU consistent with respect to the same partition of the variables $L, U \in \mathcal{V}$ then at a fixed point $\overline{\mathcal{P}}$ the complete assignment A, where $A[x] = min(dom(x))$ for $x \in L$ and $A[y] = max(dom(y))$ for $y \in U$, is a feasible solution.*

Proof. At a fixed point, filtering algorithms are no longer able to filter inconsistent values and thus local consistency is ensured to the prescribed strength of the filtering algorithms. Thus, if each constraint has a filtering algorithm that enforces LU consistency with respect to the same partition $L, U \in \mathcal{V}$ then assigning all variables in L to their lower bound and all variables in U to their upper bound satisfies each of the constraints and thus ensures a feasible solution. □

We now recount the definition of bounds consistency, as used by Lopez-Ortiz et al. (2003), and show that bounds consistency and LU consistency are incomparable, i.e., one does not necessarily imply the other. Bounds consistency is most easily defined with respect to *interval supports*. Value $v \in dom(x)$ of variable x has an interval support with respect to constraint C if there exists a tuple $t \in C$ such that $t[x] = v$ and $t[y] \in [min(dom(y)), max(dom(y))]$, for every other variable $y \in vars(C)$.

Definition 5.2.3 (bounds consistency)**.** A constraint $C \in \mathcal{C}$ is bounds consistent if for each variable $x \in vars(c)$ each of the values $min(dom(x))$ and $max(dom(x))$ has an interval support in C.

We note that LU consistency and bounds consistency are incomparable. If a filtering algorithm is LU consistent for a given partition $L, U \subseteq \mathcal{V}$ then by definition the lower bounds of each variable in L and the upper bounds of each variable in U have an interval support, i.e., all based on the same tuple. However this does not ensure the upper bounds of variables in L and the lower bounds of variables in U have interval supports, and hence LU consistency does not imply bounds consistency. Conversely, if a filtering algorithm is bounds consistent both the upper and lower bounds of each variable has an interval support. However each support may be based on a different tuple, which does not ensure for any specific partition $L, U \in \mathcal{V}$ that there exists a solution to the tuple where $t[x] = min(dom(x))$ for $x \in L$ and $t[y] = max(dom(y))$ for $y \in U$.

As interval supports in bounds consistency may be based on different tuples,

assigning any variable to their upper or lower bound may make the current lower and upper bound values inconsistent. While LU consistency is in essence one-sided, assigning all the variables to the appropriate bound is based on the same tuple and hence fixing all values simultaneously to the required bound will definitely satisfy that individual constraint.

We now define the notion of protected values and redundant constraints, and discuss how they relate to LU consistency.

Definition 5.2.4 (protected values)**.** The value $d \in dom(x)$ of variable $x \in vars(C)$ is *protected* with respect to constraint C if for any tuple $t \in C$ there exists another tuple $t' \in C$ such that $t'[x] = d$ and $t'[y] = t[y]$ for all other $y \in vars(C)$.

Observe that by definition if given some partition of variables $L, U \in \mathcal{V}$, if each value $min(dom(x))$ of variables $x \in L$ and and each value $max(dom(y))$ of variables $y \in U$ are protected, then the constraint is LU consistent with respect to that partition. The definition of *protected values* can be seen as a more general description of variable locks, which will be discussed in greater detail in Section 5.3.4.

Definition 5.2.5 (redundant constraints)**.** A constraint C is *redundant* if every value $d \in dom(x)$ of every variable in the scope of the constraint $x \in vars(C)$ is protected.

Again observe that it follows from the definition that if a constraint C is redundant then it is LU consistent for every possible partition of the variables.

5.2.3 Driving Variables

In the introduction we informally described driving variables as a subset of variables such that once assigned values and constraint propagation reaches a fixed point, a search strategy can be determined in polynomial time that will always generate a complete assignment without backtracking. To more explicitly define what is meant by determining a search strategy in polynomial time that will always generate a complete assignment without backtracking, we introduce the notion of a *sub-strategy*. The definition of a sub-strategy that follows is based on the definition of a *sub-solver* defined with respect to backdoors (Williams et al., 2003).

Definition 5.2.6 (sub-strategy)**.** A sub-strategy G takes as input a CSP, $\mathcal{P}$, and satisfies the following:

- (Trichotomy) G either rejects the input $\mathcal{P}$, determines $\mathcal{P}$ correctly as unsatisfiable, or returns a search strategy $(f^{variable}, f^{value})$ that will lead to a complete assignment without backtracking;

- (Efficiency) G can be determined in polynomial time;[1]
- (Trivial solvability) G can determine if $\mathcal{P}$ is trivially true (has no constraints) or trivially false (has a contradictory constraint); and
- (Self-reducibility) if G determines $\mathcal{P}$, then G determines every subproblem $\mathcal{P}' \subseteq \mathcal{P}$.

The difference between a sub-strategy and a sub-solver is that sub-solvers return an explicit solution instead of a back-track free search strategy. Note that if a sub-strategy can determine a complete solution explicitly then it is possible to return that complete solution in the form of a back-track free search strategy where the variable selection strategy picks any variable arbitrarily and the value selection strategy simply sets that variable to the value in the complete solution. This approach is guaranteed to be back-track free as constraint propagation cannot remove any required values as they are known not to be inconsistent. Sub-strategies are thus more general than sub-solvers as they can also reason about complete assignments based on constraint propagation and thus the local consistency rules of individual filtering algorithms, which due to the complexity of fixed point computation cannot technically be completed by sub-solvers.

Definition 5.2.7 (driving variables)**.** A nonempty subset X of the variables in a problem $\mathcal{P}$ are driving variables if for any assignment, $A : X \to D$, constraint propagation will either fail, or reach a fixed point $\overline{\mathcal{P}}_{|A}$ for which there is always a sub-strategy G.

We can now define the following theorem that relates driving variables and LU consistency.

Theorem 5.2.8. *Given a problem $\mathcal{P}$ and any assignment A over the subset of variables X, if constraint propagation reaches fixed point $\overline{\mathcal{P}}_{|A}$ and each constraint C in this sub-problem has a filtering algorithm that is LU consistent for the same partition of variables, then X constitutes a set of driving variables.*

Proof. This follows immediately from the definition of driving variables and Proposition 5.2.2. For assignment A, if constraint propagation can reach a fixed point and all constraints are LU-consistent for the same partition $L, U \in \mathcal{V}$ then a sub-strategy that will always lead to a complete assignment without backtracking is one where variables are chosen arbitrarily, the value selected for all variables $x \in L$

[1]Note that, in an effort to generalise backdoors, this definition accounts for the case where a solution can be determined in polynomial time and then returned in the form of a search strategy, where variables are set to their values in the found solution.

are then $min(dom(x))$ and the value selected for all variables in $y \in U$ is then $max(dom(y))$. □

Therefore when considering driving variables with respect to LU consistency, driving variables can be seen as the set of variables for which any assignment will result in at least one of the filtering algorithms for every constraint becoming LU consistent with respect to the same partition of variables.

5.3 Related Concepts

As discussed in the introduction there are a number of related concepts to driving variables and LU consistency in the literature. We will introduce these concepts with respect to our notation and then prove how these concepts relate to our work.

5.3.1 Function Dependencies

Functional dependencies is a concept that originated from the field of relational database theory. With respect to CP and our notation, a functional dependency is a single constraint $C \in \mathcal{C}$ whose scope can be partitioned into distinct sets, $X, Y \subseteq vars(C)$, such that once all of the variables in X are fixed all of the variables in Y become fixed through propagation of this constraint. More generally, we say that given two distinct subsets of the variables, $X, Y \subseteq \mathcal{V}$, that Y is *functionally dependent* on X if given any assignment over X either propagation determines an inconsistency or at a fixed point all variables in Y are fixed. The following proposition states that functional dependencies are a special case of driving variables.

Proposition 5.3.1. *Given a partition of the variable set $X, Y \subseteq \mathcal{V}$ where Y is functionally dependent on X, then X represents a set of driving variables*

It is possible for sets of variables to functionally dependent on each other. These situations are ubiquitous in problems that incorporate multiple viewpoints. Viewpoints are a concept first informally introduced by Geelen (1992) and then formally defined by Law and Lee (2002). Multiple viewpoints combine two or more models of the same problem and introduce functional dependencies such that assignments in one viewpoint can be translated into assignments in the others (Cheng et al., 1999). In such problems, search can be constrained to only the variables from one of the viewpoints and any feasible assignment will instantiate all of the dependent variables, and this has been show to have practical benefit (Hnich et al., 2004). Although restricting search based on functional variable dependencies has been

shown to improve practical performance, for our purposes the requirement that all remaining variables are fixed through propagation of individual constraints is too restrictive.

5.3.2 W-Cutsets and Constraint Graphs

Constraint graphs and w-cutsets are concepts introduced to demonstrate how relational consistency measures and the structure of constraints can be used to reason about complete assignments. There has been a significant amount of work in this area, for a more detailed introduction to the concepts we refer the reader to the following textbook chapters (Dechter, 2003a,b,c, 2006).

A constraint graph is a graphical representation of a problem where each node represents a variable and edges are introduced between pairs of nodes if the associated variables are both contained in the scope of at least one constraint.[2] An *ordered constraint graph* is a constraint graph with a given ordering of the nodes. Ordered constraint graphs are important with respect to backtrack free search in CP as orderings can be implemented as the variable selection strategy. The nodes adjacent to a given node x that also precede it with respect to the ordering are called its *parents*. The width of a node in an ordered graph is its number of parents. The *width* of an ordering is the maximum width of any node. The width of a graph is the minimum width over all of the possible orderings of the graph. It is possible to relate the width of the constraint graph to special types of relational consistency measures.

An assignment over a subset of the variables $X \subseteq \mathcal{V}$ is called *consistent* if it satisfies all the constraints C for which the scope of the constraint is a subset of these variables, i.e., $vars(C) \subseteq X$. A problem is *i-consistent* if for any consistent assignment over a subset of $i-1$ variables there exists an instantiation of any other variable such that the assignment over i variables is also consistent. Arc-consistency and path-consistency corresponds to special cases of i-consistency where i equals 2 and 3, respectively. A problem is *strongly i-consistent* if it is j-consistent for all $j \leq i$. In his seminal work, Freuder (1982) famously proved that if a constraint graph with has a width $i-1$, and if it is also strongly i-consistent, then there a is backtrack-free variable ordering. Furthermore, although it is NP-complete to determine whether a graph G has width at most a given variable k (Arnborg et al.,

[2] An alternative representation is based on *constraint hypergraphs* (Dechter, 2006), where the scope of each constraint is represented explicitly as a subset of variables known as a hyperedge, however for simplicity we narrow the analysis to constraint graphs and then simply comment on extensions.

1987), when k is a fixed constant the graphs with width k can be constructed in linear time (Bodlaender, 1993).

It is uncommon however that the width of the constraint graph is less than the level of strong i-consistency. Methods have been proposed to try and achieve this however by both increasing the level of strong i-consistency of the graph, and by reducing the width of the graph. The latter is based on the concept of w-cutsets. Given a constraint graph a subset of nodes is called a w-cutset if when the subset is removed the resulting graph has an width less than or equal to w. A *minimal w-cutset* of a graph has a smallest size among all w-cutsets of the graph.

Proposition 5.3.2. *If X is a w-cutset to the constraint graph and the resulting problem is strongly w+1-consistent at a fixed point, then X is also a set of driving variables.*

Proof. The proof follows immediately from the various definitions. If X is a w-cutset then any possible assignment over X results in a problem with a width of w. Given that after constraint propagation the corresponding problem is strongly w+1-consistent, we know from Bodlaender (1993) that there is a backtrack-free variable ordering that can be determined in linear time. Therefore using this ordering as the variable selection strategy and assigning those variables to any value in the corresponding domain will lead to a complete solution. □

Practically, there are a number of challenges of both determining w-cutsets and enforcing strong i-consistency. Firstly, in general determining a minimal w-cutset, and thus determining the driving variables of the problem, is intractable. Secondly, strong i-consistency is what is known as a relative consistency measure as it is based on subsets of variables being extended to larger subsets, not based on the local consistency rules of the filtering algorithms of the constraints. As a result enforcing strong i-consistency in typical CP solvers can require the addition of an exponential number of constraints (Bessiere, 2006).

5.3.3 Strong Backdoors to Typical Case Complexity

A (strong) backdoor is a subset of variables that once fixed there is a known polynomial-time algorithm, referred to as a sub-solver, that can either find a solution for the remaining variables or determine that a solution does not exist (Williams et al., 2003). Backdoors are particularly useful in understanding the excellent scaling behaviour of modern boolean satisfiability (SAT) solvers in many practical instances, which appear to defy the theoretical intractability of the problem. With respect to SAT, a backdoor is a subset of variables that once assigned

values, the problem reduces to a tractable class of problems such as 2-SAT and HornSAT. Although the definition of backdoors naturally holds for the more general Constraint Satisfaction Problem (CSP), the requirement that the solver is a polynomial-time algorithm is very restrictive for practical problems due to the complexity of reaching a fixed point.

In CP, determining a fixed point for variables with finite domains by constraint propagation in general takes pseudo-polynomial-time (Lhomme, 1993)[3]. Moreover, fixed point computation has been shown to be weakly NP-complete even when restricted to very simple constraints such as binary linear constraints (Bordeaux et al., 2011). In contrast, determining a fixed point in SAT can be achieved in linear time by unit-propagation (Zhang and Stickel, 1996). As pseudo-polynomial algorithms are technically exponential functions of their input size, in the strict sense they are not considered polynomial (Hall, 2013). Furthermore, as backdoors are defined with respect to polynomial sub-solvers and fixed points are computed in pseudo-polynomial-time, in general it is thus not possible to include constraint propagation as part of the sub-solver. As a result, given a partial assignment of the backdoor variables in CP, it is not possible to reason about the status of the domains of the remaining variables based on the local consistency enforced by the filtering algorithms of the various constraints. This is significantly limiting. Consequently, the analysis of backdoors in CP in the literature to date appear to be largely theoretical and appear to primarily focus on extensionally represented constraints, i.e., constraints for which all possible value combinations are known (Gaspers et al., 2017a,b).

Proposition 5.3.3. *If $X \subseteq \mathcal{V}$ is a strong backdoor to problem $\mathcal{P}$, then X is also a set of driving variables.*

Proof. If X is a strong backdoor to the problem $\mathcal{P}$ then for any assignment $A : X \rightarrow \mathcal{D}$ there exists a sub-solver to $\mathcal{P}_{|A}$ that returns a satisfying assignment to the remaining or concludes unsatisfiability. Due to the self-reducibility property of sub-solvers we can use the sub-solver to $P_{|A}$ as the sub-strategy to $\overline{\mathcal{P}}_{|A}$, noting $\overline{\mathcal{P}}_{|A} \subseteq \mathcal{P}_{|A}$. □

In many CSPs there are upper bounds that can be assumed on the size of the domains of the variables. In such cases the complexity of determining a fixed point is consider fixed-paramater tractable, which is accepted to be polynomial solveable. In such cases strong backdoors and driving variables and equivalent.

[3]This of course assumes that all filtering algorithms can be performed in polynomial-time.

Proposition 5.3.4. *If $X \subseteq \mathcal{V}$ is a set of driving variables, and the size of the domains of variables is bounded by some parameter k, then X is also a strong backdoor.*

Proof. The sub-solver used for the backdoor can be determined by the following three observations. Firstly, as k bounds the size of the domains of the remaining variables, constraint propagation is now polynomial with respect to the number of remaining variables. Secondly, a sub-strategy can be determined in polynomial time. Thirdly, the sub-strategy can be used to determine a complete solution without backtrack and thus constraint propagation is called at most n more times, where n is the number of remaining variables. Hence the sub-solver is has polynomial-complexity. □

However there are other problems for which the size of the domains can be very large. One area in particular where the size of domains are generally exceptionally large, or even unbounded, is in the application domain of scheduling. In the next section we will discuss driving variables and LU consistency with respect to some filtering algorithms used in constraint-based scheduling solvers.

5.3.4 Variable Locks

The notion of *variable locks* was first introduced by Achterberg (2007) to help overcome the inaccessibility of dual information about the variables when considering abstract constraints such as global constraints in CP. Informally, an *up-lock* is a constraint that blocks a variable from taking values towards its upper bound and a *down-lock* is a constraint that blocks a variable from taking values towards its lower bound. The notions of up-locks and down-locks have a clear resemblance to the definition of LU consistency that is worth making explicit.

More formally, the notion of locks are defined with respect to whether a constraint is *monotone decreasing* or *monotone increasing*. With respect to our notation, a constraint C is *monotone decreasing (increasing) in variable* x_j, if for any assignment $t \in C$, for all values in its domain less (greater) than $t[x_j]$, i.e. $v \in dom(x_j)$ where $v < t[x_j]$ ($v > t[x_j]$), there exists another feasible assignment $t' \in C$, with $t[x_k] = t'[x_k]$ for all $k \neq j$ and $t'[x_j] = v$. Simply put, for any feasible assignment it is possible to decrease or increase the value of the specific variable and ensure feasibility of the individual constraint. We say that constraint C needs to *down-lock* (*up-lock*) variable x_j if and only if the constraint is not monotone decreasing (increasing) in variable x_j.

In the following theorem, we show how up-locks and down-locks can be

used to reason about LU consistency, but in an admittedly very simple and one-directional manner.

Theorem 5.3.5. *A constraint $C \in \mathcal{C}$ is LU-consistent for the disjoint partition of variables $L, U \subseteq vars(C)$ if all variables $x \in L$ do not need a down-lock and all variables $x \in U$ do not need an up-lock.*

Proof. Let $t \in C$ be any feasible assignment to C. The proof is completed directly by constructing an assignment $t'[x]$ by changing the value of one variable at a time, where $t'[x] = min(dom(x))$ for all $x \in L$ and $t'[x] = max(dom(x))$ for all $x \in U$.

First consider any of the lower variables, $x_j \in L$. As C does not need a down-lock for x_j, C is monotone decreasing in x_j. Hence if x_j is not set to its lower bound in t, i.e., $t[x_j] \neq min(dom(x))$, then by definition there exists a feasible assignment t' such that $t'[x_k] = t[x_k]$ for all $k \neq j$ and $t'[x_j] < t[x_j]$. Replace t with t' and repeat for the remaining lower variables.

Similarly, consider any of the upper variables, $x_j \in U$. As C does not need an up-lock for x_j, C is monotone increasing in x_j. Hence if x_j is not set to its upper bound in t, i.e., $t[x_j] \neq max(dom(x))$, then by definition there exists a feasible assignment t' such that $t'[x_k] = t[x_k]$ for all $k \neq j$ and $t'[x_j] > t[x_j]$. Replace t with t' and repeat for the remaining upper variables.

As t' is a feasible assignment, the constraint is *LU-consistent* for the specific partitions L and U. □

Note the inverse of the Theorem 5.3.5 is not necessarily true, i.e., if constraint C is LU-consistent for the disjoint partition of variables $L, U \subseteq vars(C)$, this does not imply that the constraint does not need to down-lock each lower variable $x \in L$ and does not need to up-lock each upper variable $x \in U$. This comes from the fact that ensuring a specific assignment exists for a constraint is not sufficient to imply that individual variables are monotone increasing or decreasing with respect to that constraint.

The key difference between LU consistency compared with variable locks is that LU consistency is a local consistency measure and thus must reason about the feasibility of all the variables in the scope of a constraint simultaneously. In contrast, locks consider individual variables of a constraint in isolation. The following example demonstrates how LU consistency can observe additional scenarios to satisfy a constraint than those observed by simply considering the locks on individual variables.

Example 5.3.6. Consider the constraint

$$ax_1 + b \leq x_2,$$

where a and b are fixed parameters and x_1 and x_2 are variables. For x_1, the constraint needs an up-lock but not a down-lock. For x_2, the constraint needs a down-lock but not an up-lock. From the point of view of locks, it is only possible to see that setting x_1 to its lower bound and x_2 to its upper bound will satisfy the constraint. From the point of view of LU consistency however, as will be shown explicitly in Section 5.4.1, setting both variables to their lower bound, both to their upper bound, as well as the scenario observed from the perspective of locks, will all satisfy the constraint.

The manner in which locks have been used in practice is subtly different to how we utilise LU consistency in this chapter. Variable locks have been used for a range of different purposes. Achterberg (2007) show how the number of up-locks and down-locks (if any) of individual variables can be used to guide primal heuristics such rounding heuristics, diving heuristics, and improvement heuristics. Furthermore, in the case were a variable does not have any up-locks or down-locks, that the variable can be fixed in a process known as *dual fixing*. In the recent PhD thesis, Heinz (2018) investigates variable locks with respect to cumulative scheduling problems in great detail. Variable locks were used to remove certain variables from the scope of constraints and increase the strength of propagation of specific constraints under a range of different conditions.

In this chapter LU consistency is used to solely reason about the subsets of variables in a problem that can be extended to a complete solution once successfully fixed. We demonstrate how being able to reasoning about assigning subsets of variables to values at the same time is an extremely powerful component of LU consistency. By considering the assignment of variables simultaneously, the conditions resulting in LU consistency are significantly different to those that are considered by Heinz (2018) in the analysis of locks.

5.4 LU Consistency in Special Cases of Scheduling Constraints

Scheduling is one of the most successful application areas of CP (Laborie et al., 2018). The success of constraint programming in scheduling is achieved based on

the combination of a range of factors. These factors can demonstrate why it is often necessary for variables in the accompanying CSP models to have large domains, and thus the need to consider the complexity of fixed point computation explicitly. Scheduling problems can be modelled very naturally and compactly as CSPs. Typically integer variables represent the *events* (the starts and ends of intervals of time) that may need to be scheduled, the domain of the variables represent the possible times that the event can be executed, and the scheduling-specific global constraints and filtering algorithms reason over the type and amount of resources required by events, as well as the necessary precedence relations between pairs of events. The recent introduction of interval variables and their respective global constraints only further improve this modelling capability.

In scheduling problems, as in packing problems, it is often sufficient to only reason over the upper and lower bounds of the domains of variables. This is a consequence of two factors. Firstly, the vast majority of scheduling-related filtering algorithms work by simply increasing the lower bound and decreasing the upper bound of variables according to certain local consistency rules (Vilím, 2007). Secondly, many scheduling problems have what are known as *regular objective functions*. For a problem with a regular objective function it is never possible to improve a solution by only starting events later. This quality means that when branching a variable can always be set to its lower bound.

Due to the fact that both branching and filtering algorithms often only consider the bounds of variables, some solvers do not explicitly enumerate the entire domains of the variables. Instead it is sufficient to represent the domain of an integer variable simply with two natural numbers: one for the lower bound, and one for the upper bound. Filtering algorithms still work by increasing the lower bound and decreasing the upper bound, but now an inconsistency is detected if the upper bound is less than the lower bound. This representation alleviates the computational overhead of representing scheduling problems with very large time-horizons. In fact, CP Optimizer represents these variable bounds using double-precision floating-points and thus the assumed maximum bound on the size of the domains of the variables is 2^{52}. Although this bound is clearly conservative for many practical problems, it highlights the point that domains in many CSPs can be extremely large and thus the fixed point complexity in CP should not be simply dismissed.

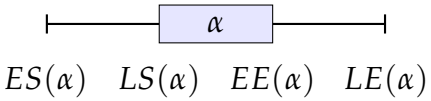
In this section we consider a number of filtering algorithms and consistency checks for existing global constraints that arise in scheduling problems. The constraints considered here are motivated by the many different precedence con-

straints, sequence variables, cumulative function expressions, and state functions incorporated in CP Optimizer. [4] In all of the constraints considered in this chapter, filtering is achieved by increasing the lower bound and decreasing the upper bound of the integer variables.

We consider interval variables $\alpha \in \texttt{interval}$ as considered by CP Optimizer. In this chapter, we assume that all interval variables must be present. The analysis presented in this chapter can be extended to consider optional interval variables but largely in the trivial case where all optional interval variables are fixed to be absent. Furthermore for shorthand let:

- the earliest start of an interval be denoted by $ES(\alpha) = min(dom(s(\alpha)))$;
- the latest start of an interval be denoted by $LS(\alpha) = max(dom(s(\alpha)))$;
- the earliest end of an interval be denoted by $EE(\alpha) = min(dom(e(\alpha)))$; and
- the latest end of an interval be denoted by $LE(\alpha) = max(dom(e(\alpha)))$.

Figure 5.1: A representation of an interval with a compulsory region.



For consistency we will use A to represent the set of interval variables and $X(A) := \bigcup_{\alpha \in A}\{s(\alpha) \cup e(\alpha)\}$ for the corresponding set of integer variables, i.e., the start and end integer variables associated with the set of interval variables A. As the filtering algorithms work by updating the bounds on integer variables, we will use the notation $min(dom(x)) \leftarrow v$, to for example state that the lower bound of variable x is updated to some value v. Furthermore, as many of the filtering algorithms consider interval variables with a compulsory region, let $Comp(A) \subseteq A$ be the set of intervals with a compulsory region, i.e., $Comp(A) := \{\alpha \in A | LS(\alpha) \leq EE(\alpha)\}$. Figure 5.1 demonstrates how an interval variable with a compulsory region is visualised as a whisker-plot, the left whisker represents the $ES(\alpha)$, the left side of the box represents the $LS(\alpha)$, the right side of the box represents the $EE(\alpha)$ and the right whisker represents the $LE(\alpha)$.

[4] In CP Optimizer, many constraints are combined to form a global constraint under the hood by the solver. For example, `alwaysEqual` and `alwaysConstant` define different interactions between individual interval variables and a state function. For each state function, CP Optimizer combines all of the individual constraints into one global constraint and has filtering algorithms on the global constraint. Equivalent techniques exist for cumulative functions and sequence variables. Hence in this chapter, we present simplified versions of these global constraints but cannot give a reference as, to our knowledge, have not been made public by the team at CP Optimizer.

5.4.1 Inequality / Precedence Constraints

Consider the binary inequality constraint between two integer variables $x, y \in X$,

$$y \geq x + a, \tag{5.1}$$

where a is a constant. Inequality constraints are prevalent in scheduling problems as they are used to represent generalised precedence relations. Filtering this constraint is achieved as follows

$$min(dom(y)) \leftarrow max(min(dom(y)), min(dom(x)) + a) \tag{5.2}$$
$$max(dom(x)) \leftarrow min(max(dom(x)), max(dom(y)) - a) \tag{5.3}$$

It is often possible to group all inequality relations into a temporal network, where nodes represent the events corresponding to integer variables and then, with respect to Equation (5.1), an arc is drawn from the predecessor x, to the successor y with arc-weight a. By grouping all inequalities into a temporal network, the propagation of Equations (5.2) and (5.3) can be achieved by using shortest path algorithms. Furthermore the structure of the temporal network can be used to help construction heuristics find initial feasible solutions. Regardless, the basic filtering rule offers a simple example of how LU consistency can be determined. First we prove partitions of the variables for which the filtering rule enforces the inequality constraint to be LU-consistent, then we prove a requirement on bounds of the variables for which the constraint is redundant.

Proposition 5.4.1. *The binary inequality constraint $y \geq x + a$ for integer variables $x, y \in X$ and constant $a \in Z$ is LU-consistent with (1) $L = \{x, y\}, U = \emptyset$, (2) $L = \emptyset, U = \{x, y\}$, and (3) $L = \{x\}, U = \{y\}$.*

Proof. From filtering rule (5.2) we know at a fixed point $min(dom(y)) \leq min(dom(x) + a)$ and thus $L = \{x, y\}, U = \emptyset$ is LU-consistent. Similarly from (5.3) we have $max(dom(x)) \leq max(dom(y)) - a$, and thus $L = \emptyset, U = \{x, y\}$ is LU-consistent. Finally since $min(dom(x)) \leq (max(dom(x))$ and from (5.3) we have $L = \{x\}, U = \{y\}$ is LU-consistent. □

Proposition 5.4.2. *The inequality constraint is redundant if $min(dom(y)) \geq max(dom(x)) + a$.*

Proof. If $max(dom(y)) \geq min(dom(y)) \geq max(dom(x)) + a \geq min(dom(x)) + a$,

hence filtering rules (5.2) and (5.3) would never be applied and constraint (5.1) would always be satisfied for any choice of x and y. □

An immediate consequence of Proposition 5.4.2 is that if one of the variables x or y are fixed, then after propagation of the constraint the constraint will become redundant.

5.4.2 Disjunctive with Setup Times

Many scheduling problems involve disjunctive resources which can only perform one activity at a time. Examples of such a resource are workers, machines and vehicles. From the point of view of the resource, a solution is a sequence of activities to be processed. Beside the fact that activities in the sequence do not overlap in time, common additional constraints on such resources are setup times or constraints on the relative position of activities in the sequence. Here we present a basic version of the disjunctive global constraint on interval variables,

$$\texttt{DisjunctiveIntervals}(A, T(\alpha)_{\alpha \in A}, M) \Leftrightarrow$$

$$e(a) + M[T(a), T(b)] \leq s(b) \vee e(b) + M[T(b), T(a)] \leq s(a) \qquad \forall a, b \in A, \wedge \tag{5.4}$$

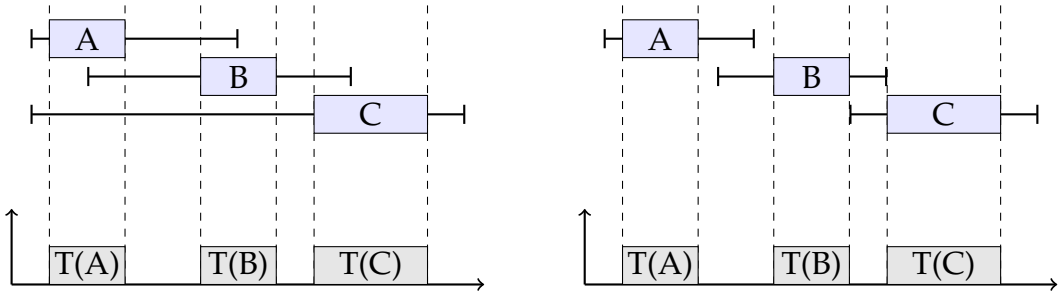
$$l^{min} \leq e(\alpha) - s(\alpha) \leq l^{max} \qquad \forall \alpha \in A \tag{5.5}$$

The input parameters are (i) a set of activities A, (ii) for each activity $\alpha \in A$ a non-negative integer is specified referred to as the type of the variable, and (iii) a transition matrix M that represents the minimal non-negative distance that must separate every two interval types in the sequence from the end of one and the start of the next. Here we assume that M obeys the triangle inequality. The constraint is satisfied if every pair is separated by the required distance (5.4) and if the length of the interval variables are allowed (5.5).

A simple filtering algorithm for the above constraint is based on a time-table. Figure 5.2 visualises how the filtering algorithm works. We will now show how the earliest starting times of intervals can be updated by considering the timetables. A symmetrical rule exists for the latest end but for simplicity we omit this.

For every compulsory interval $b \in Comp(A)$ and every interval $a \in A \setminus \{b\}$, if

Figure 5.2: The time table filtering rule for `DisjunctiveIntervals` for a small example consider three interval variables: A, B, C. The left side indicates before constraint propagation, the right side indicates the regions after constraint propagation.



$LS(b) - EE(a) < M[T(a), T(b)]$, then the following filtering rules can be enforced,

$$ES(a) \leftarrow \max(ES(a), EE(b) + M[T(b), T(a)]) \tag{5.6}$$
$$EE(a) \leftarrow \max(EE(a), ES(a) + l^{min}(a)) \tag{5.7}$$

Proposition 5.4.3. *If $EE(\alpha) > LS(\alpha)$ for all $\alpha \in A$, then the constraint is LU-consistent with (1) $L = A, U = \emptyset$ and (2) $L = \emptyset, U = A$.*

Proof. We consider the case where $L = X$ and $U = \emptyset$ and for the sake of contraction assume the constraint is not LU-consistent for this partition. Hence we assume that at a fix point where variables have non-empty domains fixing $s(\alpha) = ES(\alpha)$ and $e(\alpha) = EE(\alpha)$ for all $\alpha \in A$ violates the constraint. From Proposition 5.4.1 we know that inequalities in Equation 5.5 must be satisfied for this partition. Hence Equation 5.4 must be violated and thus there must exist a pair of interval variables $a, b \in A$ such that $ES(b) < EE(a) + M[T(a), T(b)]$ and $ES(a) < EE(b) + M[T(b), T(a)]$. This implies that $LS(b) - EE(a) \geq M[T(a), T(b)]$ otherwise filtering algorithm 5.6 would ensure $ES(a) \geq EE(b) + M[T(b), T(a)]$, which would violate the assumption that we are at a fixed point. Similarly $LS(a) - EE(b) \geq M[T(b), T(a)]$. Rearranging we show that $LS(b) \geq M[T(a), T(b)] + EE(a) > M[T(a), T(b)] + LS(a) \geq EE(b) + M[T(b), T(a)] + M[T(a), T(b)]$. As all values in the transition matrix are non-negative, this implies that $LS(b) > EE(b)$, which violates the condition that $EE(\alpha) > LS(\alpha)$ for all $\alpha \in A$. A symmetrical argument can be made for $L = \emptyset, U = X$. □

Proposition 5.4.4. *If $EE(\alpha) - LS(\alpha) \geq l^{min}(\alpha)$ for all $\alpha \in vars(C)$, then the constraint is LU-consistent with $L = e(\alpha)$ and $U = s(\alpha)$ for all $\alpha \in X$.*

Proof. From Propositions 5.4.1 and 5.4.2 the maximum and minimum length in-

equalities are LU-consistent and redundant respectively. Hence Equation 5.4 must be violated and there must exist a pair of intervals $a, b \in A$ such that $ES(b) < EE(a) + M[T(a), T(b)]$ and $ES(a) < EE(b) + M[T(b), T(a)]$. However at a fixed point either $LS(b) - EE(a) \geq M[T(a), T(b)]$ or the filtering algorithm 5.6 would ensure $EE(b) + M[T(b), T(a)] \leq ES(a) \leq LS(b)$, which is a contradiction. □

5.4.3 State Functions

It is often necessary to reason over the state of different resources throughout a schedule. Typical examples include the type of tool in a machine, the type of material present in a location, or the current temperature of an oven. Time may be required to transition from one state to another, certain activities may only be possible to be completed when the resource is in a given state. Moreover, it might be a requirement that an activity begins or ends in line with the beginning or end of a certain state. To this end, CP Optimizer, introduces the notion of a *state function* (Laborie et al., 2018).

A state function $f = ([s_\sigma, e_\sigma) : v_\sigma)_{\sigma \in \Omega}$ is a set of non-overlapping intervals where each interval $\sigma \in \Omega$ has a fixed start and end time, $[s_\sigma, e_\sigma)$, and is associated with a non-negative integer value v_σ that represents the state of the function over the segment. We define the domain of f as $D(f) = \cup_{i \in [1,n]} [s_i, e_i)$, i.e., the set of points where the state function is associated with a state. For a fixed state function f and a point $t \in D(f)$, we denote $[s(f,t), e(f,t))$ the unique segment of the function that contains t and $f(t)$ the value of this segment. A state function can also be endowed with a transition distance matrix M. If $M[v, v']$ is a transition distance between state v and state v', then $\forall \sigma \in \Omega$, $e_\sigma + M[v_\sigma, v_{\sigma+1}] \leq s_{\sigma+1}$.

Here we present a basic version of constraints on state functions over interval variables,

$$\texttt{StateIntervals}(A, (v_\alpha, algn^s_\alpha, algn^e_\alpha)_{\alpha \in A}, M) \Leftrightarrow$$

$$\exists f = ([s_\sigma, e_\sigma) : v_\sigma)_{\sigma \in \Omega}) \text{ s.t.}$$

$$[s(\alpha), e(\alpha)) \subseteq [s(f, s(\alpha)), e(f, s(\alpha))), \qquad \forall \alpha \in A \quad (5.8)$$

$$algn^s_\alpha \Rightarrow s(\alpha) = s(f, s(\alpha)), \qquad \forall \alpha \in A \quad (5.9)$$

$$algn^e_\alpha \Rightarrow e(\alpha) = e(f, e(\alpha)), \qquad \forall \alpha \in A \quad (5.10)$$

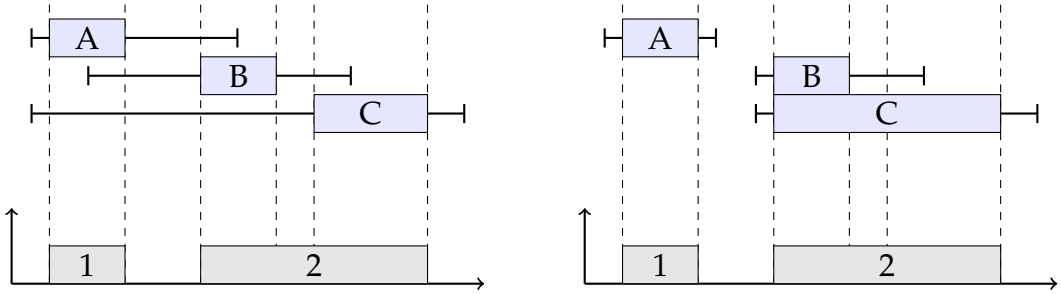
$$f(s(\alpha)) = v(\alpha), \qquad \forall \alpha \in A \quad (5.11)$$

$$e(\sigma) + M[v(\sigma), v(\sigma+1)] \leq s(\sigma+1), \qquad \forall \sigma \in \Omega \quad (5.12)$$

$$l^{min} \leq e(\alpha) - s(\alpha) \leq l^{max} \qquad \forall \alpha \in A \quad (5.13)$$

As input the constraint takes (i) a set of interval variables A, (ii) for each variable $\alpha \in A$ a triple is defined that contains the required value $v(\alpha)$ the interval can take, and two boolean values $algn^s_\alpha$ and $algn^s_\alpha$ that indicate the start and end alignment property respectively, and (iii) a transition matrix M between all the possible values. The constraint is satisfied if there exists a state function f such that: every interval variable in the scope of the constraint is contained within a single interval of the state function (5.8); if the interval is start aligned then the start time of the interval variable aligns with the start time of the corresponding interval from the state function (5.9); if the interval is end aligned then the end time of the interval variable aligns with the end time of the corresponding interval from the state function (5.10); the value of the state function in an interval is equal to value required by all the corresponding interval variables (5.11); the intervals of the state function respect the transition matrix (5.12); and the minimum and maximum lengths of the interval variables are respected (5.13).

Figure 5.3: Time Table filtering of `StateIntervals` global constraint with three intervals, A, B and C. The value required by A is 1 and the value required by B and C is 2. Interval C must be start aligned to its corresponding state interval. On the left is the bounds and tentative state function before propagation, whereas on the right is after propagation. Note the travel matrix between states is sufficient that tentative intervals for B and C are merged. The latest start bound of C is updated due to its alignment.



Tentative intervals of the state function can be inferred based on the current bounds of the interval variables with a compulsory region. Figure 5.3 illustrates this on a small example. Let $Comp(A) \subseteq A$ be the set of interval variable such that $EE(\alpha) \geq LS(\alpha)$. For every $\alpha \in Comp(A)$ a tentative interval σ is constructed with $s(\sigma) = LS(\alpha), e(\sigma) = EE(\sigma), v(\sigma) = v(\alpha)$, and $vars(\sigma) = \{\alpha\}$. As a number of the tentative intervals may be overlapping, the current state function is constructed by merging these overlapping intervals. More precisely, if two tentative intervals σ_1, σ_2 must overlap, i.e., $e(\sigma_1) + M([v(\sigma_1), v(\sigma_2)]) \geq s(\sigma_2)$ or

$e(\sigma_2) + M([v(\sigma_2), v(\sigma_1)]) \geq s(\sigma_1)$, then they are merged into a single tentative interval σ by $s(\sigma) = min(s(\sigma_1, s(\sigma_2)), e(\sigma) = max(e(\sigma_1, e(\sigma_2)), v(\sigma) = v(\sigma_1) = v(\sigma_2)$ and $vars(\sigma) = vars(\sigma_1) \cup vars(\sigma_2)$. If merging is not possible due to the required values of the intervals, then an inconsistency is determined.

The final set of tentative intervals of the state function can be used to filter domains on the interval variables. Firstly for every interval variable $\alpha \in A$ and every tentative interval of the state function $\sigma \in \Omega$ if $s(\sigma) - EE(a) < M[v(a), v(\sigma)]$ and $v(\sigma) \cap v(\alpha) = \emptyset$, then the earliest start of α can be filtered according to the following rules,

$$ES(\alpha) \leftarrow max(ES(\alpha), e(\sigma) + M[v(\sigma), v(\alpha)]) \tag{5.14}$$

This is identical to how time-tabling is used to filter the bounds of interval variables in the `DisjunctiveIntervals` constraint. Again a symmetrical rule exists for propagating the latest start and end times but is omitted here. Filtering can also be performed using the start and end alignment information. For every interval with a compulsory region $\alpha \in Comp(A)$ that is start aligned, $algn^s_\alpha = 1$, then the following filtering rules can be implemented,

$$LS(\alpha) \leftarrow s(f, LS(\alpha)) \tag{5.15}$$

$$ES(\alpha) \leftarrow max_{\alpha' \in vars(\sigma): algn^s_{\alpha'} = 1}(ES(\alpha')) \tag{5.16}$$

The latest start time of the interval variable is filtered by Equation (5.15) to the start time of the corresponding tentative interval in the state function. The earliest start time is set to the maximum earliest start time of all other interval variables and the latest start time can then be updated if necessary by Equation (5.16).

Proposition 5.4.5. *If $EE(\alpha) - LS(\alpha) \geq l^{min}(\alpha)$ for all $\alpha \in A$, then the constraints on a state function are LU-consistent with $L = e(\alpha), U = s(\alpha)$.*

Proof. For the sake of contradiction assume the partition is not LU-consistent. From Propositions 5.4.1 and 5.4.2, the maximum and minimum length inequalities are LU-consistent and redundant respectively. Thus there must not exist a state function such that constraints (X-Y) hold. As all intervals have a compulsory region $Comp(A) = A$, the set of tentative intervals $\overline{\Omega}$ contains all the variables, i.e., $\bigcup_{\sigma \in \overline{\Omega}} vars(\sigma) = A$. Thus based on how the tentative intervals are constructed

constraints 5.8 and 5.11 must be true. Constraints (5.9) must be true otherwise filtering rule 5.15 must propagate which violates the assumption that we are at a fixed point. Symmetrically 5.16 must hold. This leaves only constraint 5.12 to be violated, however this is a contradiction as if there exists a pair of consecutive intervals $\sigma, \sigma+1 \in \Omega$ such that $\sigma + M[v(\sigma), v(\sigma+1)] > s(\sigma+1)$ they would be merged into a single tentative interval. □

Finally we formalise two reasonably trivial propositions that account for situations where the `StateIntervals` constraint ensures that some values of variables are protected. This is based on the insight that if all interval variables in the state function either require the same value or do not require any alignment, then the constraint is redundant.

Proposition 5.4.6. *If all variables $\alpha \in A$ of a state intervals constraint require the same value v and no interval requires alignment, i.e. $algn^e_\alpha = algn^s_\alpha = 0$, then the constraint is redundant.*

Proof. This comes directly from observing that all filtering is performed based on either the compulsory regions of intervals requiring different values (5.14) or from the same value but requiring alignment (5.15) and (5.16). □

The previous proposition only really helps to identify that there are situations where the `StateIntervals` constraint has no purpose. However this concept can be applied to each interval variable to show that locally some specific values in their domain might be protected. Firstly, let $B(s, e, v)$ be the subset of interval variables in A that might be affected by adding an interval to the state function from s to e with value v. More explicitly, let $B = \{b \in A | [ES(b), LE(b)) \cap [s - M[v(b), v], e + M[v, v(b)]) \neq \emptyset\}$.

Proposition 5.4.7. *For interval variable α not requiring start or end alignment and let $\bar{s} \in [ES(\alpha), LS(\alpha))$ and $\bar{e} \in [EE(\alpha), LE(\alpha))$ be possible start and end times, respectively. If all intervals $b \in B(\bar{s}, \bar{e}, v(\alpha))$ require the same value $v(b) = v(\alpha)$ and also do not require start or end alignment, $algn^s_b = algn^e_b = 0$, then $\bar{s}$ and $\bar{e}$ are protected.*

Proof. Let $t \in C$ be a solution to the constraint. The proof is completed by constructing a new solution $t' \in C$ such that $t'[x] = t[x]$ for all $x \in X(A \setminus \{\alpha\})$ and $t'[s(\alpha)] = \bar{s}$ and $t'[e(\alpha)] = \bar{e}$. As $t \in C$ we know that there exists a state function $f(t) = ([s_\sigma, e_\sigma) : v_\sigma)_{\sigma \in \Omega}$ such that (5.8-5.12) is true and that the interval inequalities 5.13 are also true. As t' is constructed from t at a fixed point we immediately get that 5.13 remains true. We construct a state function f' from f by

adding the additional tentative interval σ constructed for α to the intervals Ω of the state function f of t. If there does not exist another interval $\sigma' \in \Omega$ such that $e(\sigma) + M([v(\sigma), v(\sigma')]) \geq s(\sigma')$ or $e(\sigma') + M([v(\sigma'), v(\sigma)]) \geq s(\sigma)$, then σ is not merged with any other intervals and constraints (5.8-5.12) remain true. If there does exist such a σ, then by the definition of $B(s, e, v)$ we know that for all variables $b \in vars(\sigma)$, $v(b) = v(a)$, and $algn_b^e = algn_b^s = 0$ and thus constraints (5.8-5.12) remain true. □

5.4.4 Cumulative Expressions

Many scheduling problems must reason about the quantity of certain types of resources over time and must ensure that this quantity never exceeds given capacities. Here we present a version of such a global constraint where an interval variable can either increase or decrease the quantity of resource available at both the start and the end of the resource and the capacity of the resource may vary over time, inspired by cumulative functions in CP Optimizer. More explicitly,

$$\texttt{ReservoirIntervals}(A, (h_{s(\alpha)}, h_{e(\alpha)})_{\alpha \in A}, C_t) \Leftrightarrow$$

$$\sum_{x \in X(A): x \leq t} h_x \leq C_t \qquad \forall t \in H \tag{5.17}$$

$$l^{min} \leq e(\alpha) - s(\alpha) \leq l^{max} \qquad \forall \alpha \in A \tag{5.18}$$

As input, the constraint takes (i) a set of interval variables A, (ii) for each variable $\alpha \in A$ a tuple containing $h_{s(\alpha)}, h_{s(\alpha)} \in \mathcal{Z}$ that represents the amount of resource added at the start and end of the interval respectively, and (iii) the maximum capacity C_t of resource that can be used at time $t \in H$. The constraint is satisfied if (5.17) the resource limit never exceeds the maximum capacity at any time point and (5.18) the minimum and maximum lengths of the intervals are respected.

The typical `Cumulative` constraint is a special case of the `ReservoirIntervals` where C_t is constant for all $t \in H$, the length of the interval is fixed $l_\alpha^{min} = l_\alpha^{max}$ for all $\alpha \in A$, resource is claimed at the start $h_{s(\alpha)} \geq 0$ and the same quantity is released at the end $h_{e(\alpha)} = -h_{s(\alpha)}$ for all $\alpha \in A$.

Here we only present a consistency check that can be used on the constraint to ensure that the constraint is still potentially feasible based on the current bounds on the domains of the interval variables. We do not present any additional filtering

rules as to our knowledge they do not lead to any noteworthy cases of LU consistency. Figure 5.4 provides an illustrative example of how the minimum capacity profile is determined.

Firstly let $X^+, X^- \subseteq X(A)$ be the subset of integer variables the produce and consume resource respectively. More precisely,

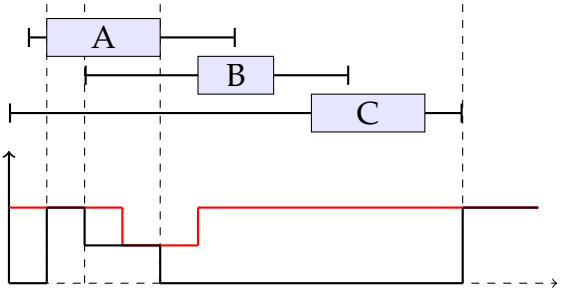
$$X^+ := \bigcup_{\alpha \in A : h^s_\alpha > 0} s(\alpha) \cup \bigcup_{\alpha \in A : h^e_\alpha > 0} e(\alpha)$$
$$X^- := \bigcup_{\alpha \in A : h^s_\alpha < 0} s(\alpha) \cup \bigcup_{\alpha \in A : h^e_\alpha < 0} e(\alpha)$$

To check if the `ReservoirInterval` constraint can still possibly be feasible, a minimum capacity profile can be inferred based on the bounds of the intervals. The constraint is still potentially feasible if all integer variables in X^+ start as late as possible and X^- start as early as possible, i.e.,

$$\sum_{x \in X^+ : max(dom(x)) \leq t} h_x + \sum_{x \in X^- : min(dom(x)) \leq t} h_x \leq C^{max}_t, \quad \forall t \in H \tag{5.19}$$

The consistency check described by constraints (5.19) can be improved. For example, consider the case where an interval variable $\alpha \in A$ represents the use of a renewable resource, i.e., $h_{s(\alpha)} = -h_{e(\alpha)} > 0$ and if the earliest end time is less than the latest start time the existence of the interval will in fact reduce the minimum capacity profile instead of increasing it. However, for the purpose of this section, 5.19) is sufficient.

Figure 5.4: The minimum capacity profile associated with a constraint with $A = [a, b, c]$ associated with tuples $(2, -1), (-1, 0), (0, 2)$ respectively, and $C_t = 2$ for all $t \in H$ except $C_t = 1$ for $t \in [3, 5)$. The relevant bounds, namely $LS(a), ES(b), EE(a), LE(c)$ are represented by a dashed line, the capacity is the red line



Before considering LU consistency, we introduce the following subsets of interval variables. We denote $A^{\sim\sim}$, where $\sim \in \{+, -, =\}$, to represent the set of interval variables depending on the amount produced or consumed at the start and end of the activities. More precisely,

$$
\begin{aligned}
A^{++} &:= \{\alpha \in A | h^s_\alpha > 0 \wedge h^e_\alpha > 0\} \\
A^{+-} &:= \{\alpha \in A | h^s_\alpha > 0 \wedge h^e_\alpha < 0\} \\
A^{+=} &:= \{\alpha \in A | h^s_\alpha > 0 \wedge h^e_\alpha = 0\} \\
A^{-+} &:= \{\alpha \in A | h^s_\alpha < 0 \wedge h^e_\alpha > 0\} \\
A^{--} &:= \{\alpha \in A | h^s_\alpha < 0 \wedge h^e_\alpha < 0\} \\
A^{-=} &:= \{\alpha \in A | h^s_\alpha < 0 \wedge h^e_\alpha = 0\} \\
A^{=+} &:= \{\alpha \in A | h^s_\alpha = 0 \wedge h^e_\alpha > 0\} \\
A^{=-} &:= \{\alpha \in A | h^s_\alpha = 0 \wedge h^e_\alpha < 0\}
\end{aligned}
$$

Note that here $X(A^{==})$ is ignored as we can assume without loss of generality that all interval variables considered by the `ReservoirIntervals` constraint have at least some contribution of the resource profile. The start and end integer variables can then be partitioned into those that will be set to their upper bound $X^{\rightarrow}$ and those that will be set to their lower bound $X^{\leftarrow}$, as follows.

$$
\begin{aligned}
X^{\rightarrow} &:= X(A^{++}) \cup X(A^{+=}) \cup X(A^{=+}) \cup \bigcup_{\alpha \in A^{+-}} s(\alpha) \cup \bigcup_{\alpha \in A^{-+}} e(\alpha) \\
X^{\leftarrow} &:= X(A^{--}) \cup X(A^{-=}) \cup X(A^{=-}) \cup \bigcup_{\alpha \in A^{-+}} s(\alpha) \cup \bigcup_{\alpha \in A^{+-}} e(\alpha)
\end{aligned}
$$

The following Proposition can now be stated.

Proposition 5.4.8. *if (1) $EE(\alpha) - LS(\alpha) \geq l^{min}(\alpha)$ for all $\alpha \in A^{+-}$, (2) $LE(\alpha) - ES(\alpha) \leq l^{max}(\alpha)$ for all $\alpha \in A^{-+}$, then the constraint is LU-consistent with $L = X^{\leftarrow}$ and $U = X^{\rightarrow}$.*

Proof. At a fixed point, the consistency check (5.19) must hold otherwise we know the constraint is infeasible. As $X^+ \subseteq U$ and $X^- \subseteq L$, then the minimum capacity profile will not change after the partial assignment is extended to a complete solution and thus (5.17) holds. It remains to show that length inequalities (5.18) are valid. This comes directly from noting that the start and end integer variables associated with interval variables from A^{++}, $A^{+=}$, $A^{=+}$, A^{--}, $A^{-=}$, $A^{=-}$ are either both in L or both in U and are thus futile. Furthermore, as it is given that

$EE(\alpha) - LS(\alpha) \geq l_{\alpha}^{min}$ for all $\alpha \in A^{+-}$ and $s(\alpha) \in U$ and $e(\alpha) \in L$, the associated minimum length is valid. A similar argument can be made for the A^{-+} interval variables. □

5.5 Case Study - A Liquid Carrying Robot Problem

5.5.1 Problem Description

The case study considers a version of the LHRSP presented in Chapter 4 of this thesis that contains a number of added real-world complexities seen in the motivating industrial problem that. More explicitly, the new complexities consider multiple methods for transferring different types of chemical as well as the fact that the pipette must be washed at a specific location before coming in contact with different type chemicals. We denote the problem in this chapter the LHRSP+.

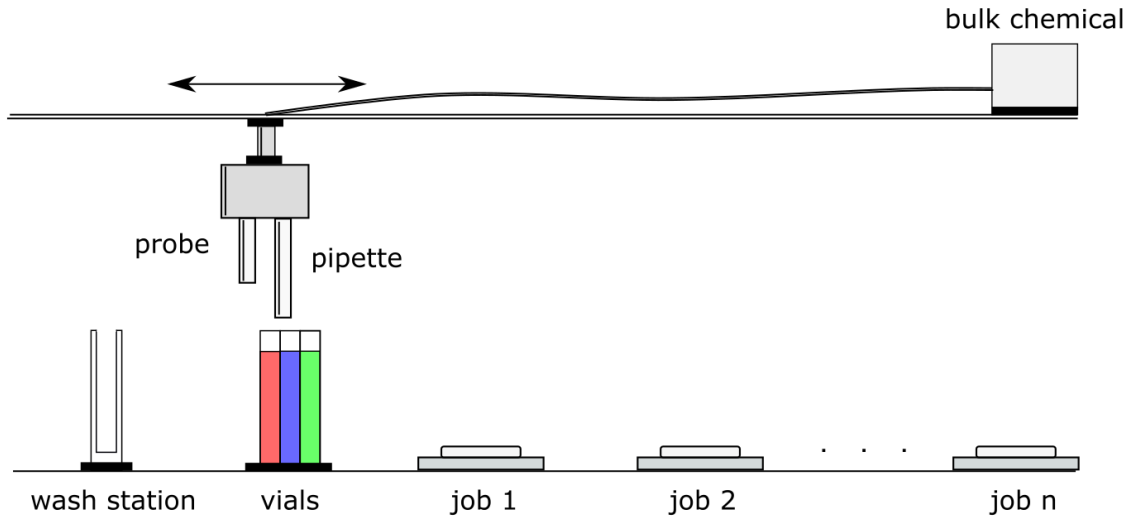
The problem considers a single robot that is responsible for processing operations O_j from a set of jobs $j \in J$ by transferring a set of chemicals C in a given manner. The robot has two methods of transferring the chemical: (1) a single pipette with finite volume, L, similar to that from Chapter 4, that it uses to transfer *high-value chemicals* from vials to the jobs and (2) a probe that is directly connected to a set of chemicals referred to as *bulk chemicals*.

Each job $j \in J$ has N_j operations. Let $O_j := \{1, 2, ..., N_j\}$ be the set of operations of job $j \in J$. Let C be the set of chemicals. Each operation corresponds to dispensing $q_{i,j}$ units of a certain chemical $c_{i,j} \in C$. For each pair of consecutive operations $i, i+1 \in J$ from the same job there exist minimum and maximum time lags, denoted by $\delta_{i,j}^{min}$, $\delta_{i,j}^{max}$ respectively.

A simple schematic of the problem is given in Figure 5.5. The schematic shows the jobs arranged in the system as well as a wash station at location index 0, where the robot can wash the pipette to avoid cross contamination, and a set of chemical vials at location 1, which contains the different high-value chemicals that can be transferred using the pipette. The schematic also shows the set of bulk chemicals and how they are directly connected by tubing to the probe of the robot.

To transfer a high value chemical to job $j \in J$ the robot must (1) move to the vials, (2) aspirate chemical into the pipette, (3) travel to job j at location index $j+1$, and (4) dispense the chemical. Assuming the required quantities of chemical do not exceed the capacity of the probe, it is possible for the robot to aspirate chemical for multiple dispenses of the same high-value chemical at the same time using the

Figure 5.5: A schematic for the problem considered in the case study. The robot has two mechanisms for transferring chemicals to jobs. The first is a pipette with finite capacity that it uses to aspirate high-value chemical from vials. The pipette must be washed at the wash station to avoid cross contamination. The second is a probe that is directly connected to a set of bulk chemicals. It is possible to dispense bulk chemical from the probe while there is high value chemical in the pipette.



pipette and then dispense the chemical accordingly. To avoid cross-contamination the robot must wash the pipette at the wash station before coming into contact with a different chemical. The robot cannot dispense bulk and high-value chemical at the same time.

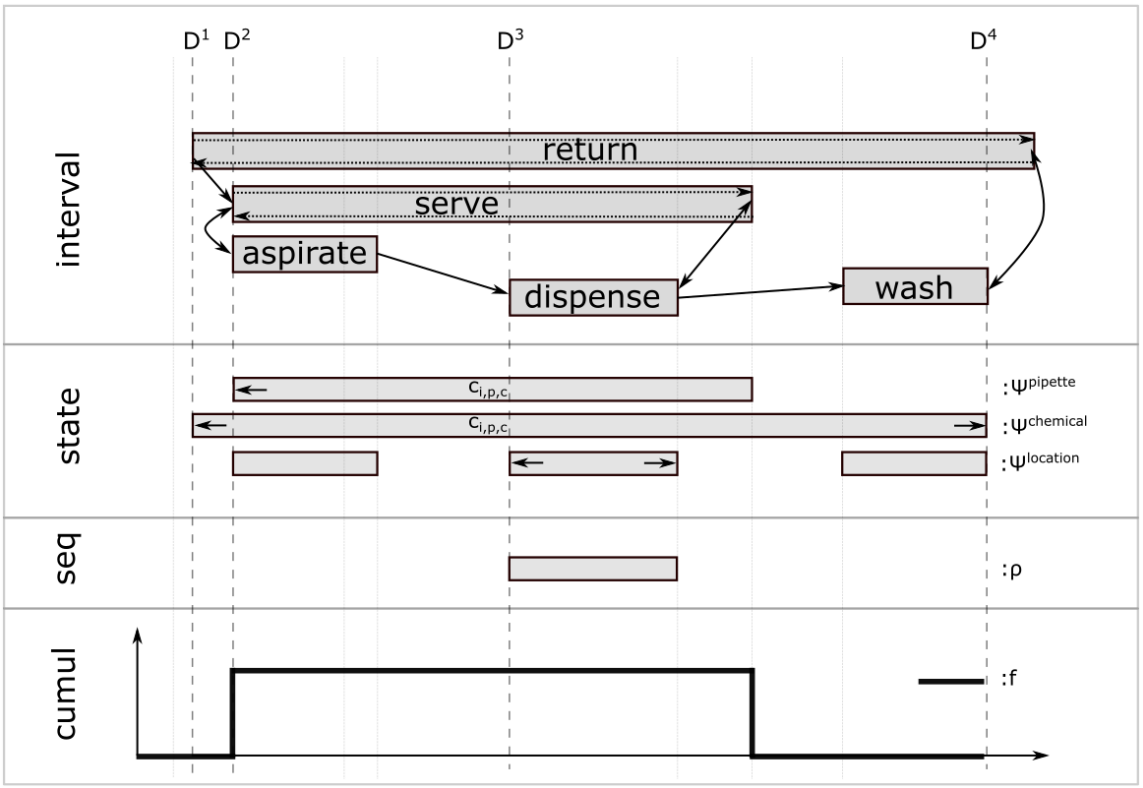
To transfer a bulk chemical the robot simply travels directly to the appropriate job and dispenses the chemical using the probe. It is possible to dispense a bulk chemical through the probe while holding high-value chemical in the pipette. Furthermore the robot does not need to clean the probe between dispenses different bulk chemicals. Thus from the point of view of scheduling all the bulk chemicals can be thought of as a single chemical. Thus we reserve chemical index 0 in the set of chemicals $C = \{0, 1, ..., n_c\}$ for bulk chemicals and the remaining indices are for the n_c high-value chemicals.

The robot takes $p^{\uparrow}$ time units to aspirate, $p^{\downarrow}$ to dispense, p^{wash} to wash the pipette, and $p^{\rightarrow}_{l,l'}$ to travel between locations $l, l' \in L$. At the start of the problem we assume the robot starts at the chemical vials and the pipette is uncontaminated. The objective is to process all the jobs in the least amount of time, i.e., minimise makespan.

5.5.2 Model Description

The model considered here is based on CP3 from Chapter 4 due to its excellent performance for the more basic LHRSP. First we will introduce the model with respect to the variables and constraints used in CP Optimizer, then comment about how this relates to the global constraints studied in this chapter. The model will be described with reference to Figure 5.6.

Figure 5.6: Visualisation of the model.



The model has five types of interval variables, three state functions, a sequence variable, and a cumulative function. Each operation has a dispense variable, $\alpha_{i,j}^{dispense}$, which corresponds to dispensing of both the bulk and high-value chemicals. Each operation that requires a high-value chemical has an additional four interval variables: (1) an aspirate variable, $\alpha_{i,j}^{aspirate}$, representing the time to aspirate the chemical at the vials, (2) a wash variable, $\alpha_{i,j}^{wash}$, representing the time to wash the pipette at the wash station, (3) a serve variable, $\alpha_{i,j}^{serve}$, representing the time from when the chemical is aspirated at the vials to the earliest time required to return to the vials after the chemical is dispensed, and (4) a return variable, $\alpha_{i,j}^{return}$, representing the time from the instant the pipette becomes contaminated by the required chemical until the chemical is eventually washed and the robot returns

to the vials to aspirate a different chemical. The key difference between the serve and return variables is that the serve variables are used to keep track of when specific units of chemical are in the pipette, whereas the return variables are used to keep track of when the pipette is contaminated. Now that washing of the pipette is considered, these times are no longer the same.

Three state functions are defined. The first ψ^{loc} represents the location of the robot. The second $\psi^{pipette}$ represents the type of chemical that is currently in the pipette. Whereas the third $\psi^{chemical}$ represents the chemical that has currently contaminated the pipette if required. The pipette state is used to align the serve variables correctly, whereas the chemical state is used to align the return variable correctly.

To ensure the dispense intervals do not overlap a sequence variable, $\rho^{location}$, is defined that only contains the dispense variables and the type of the variables corresponds to the location of the jobs. Finally a single cumulative function, $f^{capacity}$, is used to represent the amount of chemical in the pipette. A constraint on the function represents the capacity and ensures that it is never exceeded.

The full CP model can now be expressed as follows.

$$\begin{aligned}
\text{min.} \quad & max_{(i,j)\in V}(\texttt{endOf}(\alpha_{i,j}^{dispense})) && && \text{(5.20a)}\\
\text{s.t.} \quad & \texttt{startBeforeStart}(\alpha_{i,j}^{dispense}, \alpha_{i+1,j}^{dispense}, \bar{\delta}_{i,j}^{min}) && \forall (i,j) \in O && \text{(5.20b)}\\
& \texttt{startBeforeStart}(\alpha_{i+1,j}^{dispense}, \alpha_{i,j}^{dispense}, -\delta_{i,j}^{max}) && \forall (i,j) \in O && \text{(5.20c)}\\
& \texttt{startBeforeStart}(\alpha_{i,j}^{return}, \alpha_{i,j}^{serve}) && \forall (i,j) \in \overline{O} && \text{(5.20d)}\\
& \texttt{startAtStart}(\alpha_{i,j}^{serve}, \alpha_{i,j}^{aspirate}) && \forall (i,j) \in \overline{O} && \text{(5.20e)}\\
& \texttt{endBeforeStart}(\alpha_{i,j}^{aspirate}, \alpha_{i,j}^{dispense}, p_{1,j}^{\rightarrow}) && \forall (i,j) \in \overline{O} && \text{(5.20f)}\\
& \texttt{endAtEnd}(\alpha_{i,j}^{dispense}, \alpha_{i,j}^{serve}, p_{j,1}^{\rightarrow}) && \forall (i,j) \in \overline{O} && \text{(5.20g)}\\
& \texttt{endBeforeStart}(\alpha_{i,j}^{dispense}, \alpha_{i,j}^{wash}, p_{1,j}^{\rightarrow}) && \forall (i,j) \in \overline{O} && \text{(5.20h)}\\
& \texttt{endAtEnd}(\alpha_{i,j}^{wash}, \alpha_{i,j}^{return}, p_{0,1}^{\rightarrow}) && \forall (i,j) \in \overline{O} && \text{(5.20i)}\\
& \texttt{alwaysEqual}(s^{location}, \alpha_{i,j}^{wash}, 0, \leftrightarrow), && \forall (i,j) \in \overline{O} && \text{(5.20j)}\\
& \texttt{alwaysEqual}(s^{location}, \alpha_{i,j}^{aspirate}, 1, \leftrightarrow), && \forall (i,j) \in \overline{O} && \text{(5.20k)}\\
& \texttt{alwaysEqual}(s^{location}, \alpha_{i,j}^{dispense}, j+1, \leftrightarrow), && \forall (i,j) \in \overline{O} && \text{(5.20l)}\\
& \texttt{alwaysEqual}(s^{pipette}, \alpha_{i,j}^{serve}, c_{i,j}, \leftarrow), && \forall (i,j) \in \overline{O} && \text{(5.20m)}\\
& \texttt{alwaysEqual}(\psi^{chem}, \alpha_{i,j}^{return}, c_{i,j}, \leftrightarrow), && \forall (i,j) \in \overline{O} && \text{(5.20n)}\\
& \texttt{noOverlap}(\rho, M) && && \text{(5.20o)}\\
& f^{cap} = \sum_{(i,j)\in V: c_{i,j}>0} \texttt{pulse}(\alpha_{i,j}^{serve}, q_{i,j}) \leq L && && \text{(5.20p)}
\end{aligned}$$

$$\texttt{endBeforeStart}(\alpha^{return}_{i,j}, \alpha^{return}_{i',j}, p^{\rightarrow}_{0,1}) \qquad \forall (i,i',j) \in Z^{\neq} \qquad (5.20q)$$

$$\texttt{startBeforeStart}(\alpha^{return}_{i,j}, \alpha^{return}_{i',j}) \qquad \forall (i,i',j) \in Z^{=} \qquad (5.20r)$$

$$\texttt{startBeforeStart}(\alpha^{serve}_{i,j}, \alpha^{serve}_{i',j}) \qquad \forall (i,i',j) \in Z^{=} \qquad (5.20s)$$

$$\alpha^{dispense}_{i,j} \in \texttt{interval}(comp, p^{\downarrow}) \qquad \forall (i,j) \in O \qquad (5.20t)$$

$$\alpha^{aspirate}_{i,j} \in \texttt{interval}(comp, p^{\uparrow}) \qquad \forall (i,j) \in \overline{O} \qquad (5.20u)$$

$$\alpha^{wash}_{i,j} \in \texttt{interval}(comp, p^{wash}) \qquad \forall (i,j) \in \overline{O} \qquad (5.20v)$$

$$\alpha^{serve}_{i,j}, \alpha^{return}_{i,j} \in \texttt{interval}(comp) \qquad \forall (i,j) \in \overline{O} \qquad (5.20w)$$

$$\psi^{pipette}, \psi^{chemical} \in \texttt{state} \qquad (5.20x)$$

$$\psi^{loc} \in \texttt{state}(M) \qquad (5.20y)$$

$$\rho^{seq} \in \texttt{sequence} \qquad (5.20z)$$

$$f^{cap} \in \texttt{cumul} \qquad (5.20aa)$$

The objective (5.20a) is to minimise the largest end time from all of the dispense activities. Constraints (5.20b) and (5.20c) represent the minimum and maximum time lags between consecutive operations of the same job, respectively. The minimum time lags can be strengthened, similar to that in Section 4.3.1 in the LHRSP, as follows

$$\overline{\delta}^{min}_{i,j} = \begin{cases} \max(\delta^{min}_{i,j}, p^{\downarrow} + p^{\rightarrow}_{j+1,0} + p^{wash} + p^{\rightarrow}_{0,1} + p^{\uparrow} + p^{\rightarrow}_{1,j+1}) & \text{if } c_{i,t} \neq c_{i+1,j} \wedge c_{i,j}, c_{i+1,j} > 1 \\ \delta^{min}_{i,j}, & \text{Otherwise} \end{cases}$$

Constraints (5.20d)-(5.20i) are all precedence constraints between intervals of the same operation and are ordered from left to right in Figure 5.6. Constraints (5.20d) ensure the start of the return interval occurs before the start of the serve variable. Constraints (5.20e) ensures the serve and aspirate intervals start at the same time. Constraints (5.20f) ensure the start of the dispense can only occur after the end of the corresponding aspirate interval with enough time to travel from the vials to the appropriate job. Constraints (5.20g) ensure the end of the serve interval occurs exactly the amount of time for the robot to travel from the job back to the vials after the end of the appropriate dispense interval. It would still be valid to have the serve intervals end at the end of the dispense intervals. However as the serve intervals are used to track the amount of chemical in the pipette and chemical can only be aspirated at the vials, it is possible to extend the serve variables in the proposed way to pass more information to the cumulative function. Constraints (5.20h) ensure the start of the wash interval can only occur after the end of the

corresponding dispense interval with enough time to travel from the job to the wash station. Finally constraints (5.20i) ensure the end of the return interval occurs exactly the amount of time for the robot to travel from the wash station to the vials after the end of the wash interval.

Constraints (5.20j)-(5.20n) relate the state functions to the interval variables. Constraints (5.20j-5.20l) ensure that the location of the wash variables occur at the wash station, the location of the aspirates occur at the vials, and the location of the dispenses occur at the appropriate job. All of these interval variables are left and right aligned to the intervals of the location state function. Constraints (5.20m) ensure the interval of the pipette state function is always equal to the value corresponding to the appropriate high-value chemical required by the operation for the entirety of the appropriate serve variable and that these intervals are start aligned. Constraints (5.20n) ensure the interval of the chemical state function is always equal to the value corresponding to the appropriate high-value chemical required by the operation for the entirety of the appropriate return variable and that these intervals are both state and end aligned. Constraint (5.20o) ensures the intervals in the sequence variable do not overlap and are separated by the transition distance matrix M. Finally constraint (5.20p) ensures the capacity of the pipette is never exceeded.

Finally constraints (5.20q)-(5.20s) are strengthening constraints, which provide additional information to the underlying temporal network. Let $Z^{\neq}$ be the set of consecutive operations from the same job that require different high-value chemicals. Constraints (5.20q) ensure that the pipette must be washed after the dispense of the predecessor before aspirating the appropriate chemical for the successor. Likewise, let $Z^{=}$ be the set of consecutive operations from the same that require the same high-value chemical. Hence constraints (5.20r) and (5.20s) ensure that the serve and return variables of the predecessor activity occur not after the serve and return interval variables, respectively, of the successor activity. Constraints (5.20t)-(5.20aa) define the various interval variables, state functions, sequence variable and cumulative function.

5.5.3 Driving Variables of the Problem

It is possible to prove that a subset of the variables can be used as the driving variables of the problem. Firstly to clarify we note the functional dependencies implied by the precedence constraints and fixed lengths of some intervals. As displayed in Figure 5.6, the various integer start and end variables from intervals

of the same operation can be partitioned into four subsets $D^1, D^2, D^3, D^4 \subseteq \mathcal{V}$. We refer to these subsets as datums as once any variable in the datum is fixed then all other variables from the corresponding activity also become fixed through propagation. The datums are as follows:

- $D^1 := \bigcup_{(i,j)\in V: c_{i,j}>0} s(\alpha_{i,j}^{return})$;
- $D^2 := \bigcup_{(i,j)\in V: c_{i,j}>0} (s(\alpha_{i,j}^{serve}), s(\alpha_{i,j}^{aspirate}), e(\alpha_{i,j}^{aspirate}))$;
- $D^3 := \bigcup_{(i,j)\in V} (s(\alpha^{dispense}), e(\alpha^{dispense})) \cup \bigcup_{(i,j)\in V: c_{i,j}>0} e(\alpha_{i,j}^{serve})$; and
- $D^4 := \bigcup_{(i,j)\in V: c_{i,j}>0} (s(\alpha_{i,j}^{wash}), e(\alpha_{i,j}^{wash}), e(\alpha_{i,j}^{return}))$.

The following theorem can then be proven with respect to these datums.

Theorem 5.5.1. *The variables representing the start time of the dispense intervals,* $X := \bigcup_{\alpha\in V} s(\alpha^{dispense})$*, are a set of driving variables.*

Proof. The proof is based on the fact that for any assignment $A : X \to D$ if a fixed point can be reached all constraints become LU consistent for a given partition of the variables, where $L = D^4$ and $U = D^1 \cup D^2$. This can be shown by considering the constraints and objective function individually.

- Firstly D^3 is functionally dependent on X, by constraints (5.20t) and (5.20g), and hence all variables in D^3 are fixed at a fixed point.
- We then note that the scope of constraints (5.20b,5.20c, 5.20t, 5.20g, 5.20o) and the objective function (5.20a) are subsets of variables in D^3 hence must be satisfied.
- As precedence constraints (5.20f, 5.20h) contain one fixed variable from Proposition 5.4.2 we know that at a fixed point these constraints are redundant.
- For inequalities (5.20u, 5.20u, 5.20d, 5.20e, 5.20i, 5.20r, 5.20s), as both variables are either in L or both variables are in U from Proposition 5.4.1 we know that they are LU consistent.
- By Proposition 5.4.1, as precedence constraint 5.20q has the predecessor in L and successor in U, then the constraint is LU-consistent.
- Each state function corresponds to a global `StateIntervals` constraints from Section 5.4. Constraint 5.20m corresponds to the global constraint $\texttt{StateIntervals}(\bigcup_{i,j\in V: c_{i,j}>0} \alpha_{i,j}^{serve}, \bigcup_{(i,j)\in V: c_{i,j}>0}(c_{i,j}, 1, 0), \emptyset)$, and constraint (5.20n) corresponds to $\texttt{StateIntervals}(\bigcup_{i,j\in V: c_{i,j}>0} \alpha_{i,j}^{return}, \bigcup_{(i,j)\in V: c_{i,j}>0}(c_{i,j}, 1, 1), \emptyset)$, where $\emptyset$ indicates that no travel time is required.

- From precedence constraints (5.20d-5.20i), we know at a fixed point $EE(\alpha_{i,j}^{serve}) - LS(\alpha_{i,j}^{serve} \geq l^{min}(\alpha) = 0$ for each serve interval and hence (5.20m) is LU consistent for the given partition by Proposition 5.4.5. Likewise, for the return intervals and thus constraints (5.20n) is also LU consistent for the partition. Constraint (5.20p) is equivalent to the global constraint $\texttt{ReservoirIntervals}(\bigcup_{(i,j)\in V:c_{i,j}>0} \alpha_{i,j}^{serve}, (q_{i,j}, -q_{i,j})_{(i,j)\in V:c_{i,j}>0}, L)$.
- With respect to Proposition 5.4.8, A^+ refers to all the serve variables, which we know have a non-negative minimum length again due to the precedence constraints, hence again we can show that (5.20p) is LU consistent for the given partition.
- Constraints (5.20j-5.20l) correspond to $\texttt{StateIntervals}(\bigcup_{i,j\in V:c_{i,j}}(\alpha_{i,j}^{dispense}, \alpha_{i,j}^{aspirate}, \alpha_{i,j}^{wash}, T, \bigcup_{l,l'\in L} p_{l,l'}^{\rightarrow})$ where T is the set of tuples which for the dispense variables are $(j+1, 1, 1)$, the wash variables are $(0, 0, 0)$ and the aspirate variables are $(1, 0, 0)$.
- From constraints (5.20n), the precedence constraints, and Proposition 5.4.7, we know that the earliest start and earliest end of all the α^{wash} variables are protected. Likewise we know that the latest end and latest completion times for the $\alpha^{aspirate}$ are protected. Therefore (5.20j-5.20l) are LU-consistent.

□

5.5.4 Computational Results

A brief computational experiment is completed using small modifications to the data in Chapter 4. The same 24 instances where used. The washing time was set to always equal to dispense time, i.e., $p^{wash} = p^{\downarrow}$. As now some instances become infeasible, the maximum time lags were also increased by p^{wash}. In line with how the problem has been modelled, we assume the chemical with the lowest index are bulk chemicals.

The same model was solved by CP Optimizer using four different search phases. The first search phase, denoted *None*, does not define any explicit search phase, allowing CP Optimizer to perform its default search. The second search phase, denoted *Low*, asserts that the aspirate, dispense, and wash variables must be fixed before the serve and return variables are considered. The third search phase, denoted *High*, asserts that the dispense variables must be fixed before the remaining interval variables are considered. Finally the search phase, denoted *Complete*, again asserts that the dispense variables must be fixed, then it asserts that

the integer variables from datums D^1 and D^2 are fixed to their upper bounds, and variables from datum D^4 are set to their lower bounds. As CP Optimizer does not currently allow search phases to set interval variables to their upper bounds, additional integer variables are required for the start of the return and serve variables in order to implement the *Complete* search phase correctly.

The results after 1 minute wall time are summarised in Table 5.1. The precedence-based and resource-based lowerbounds are denoted *LB-T* and *LB-R*, respectively. The results are consistent with the additional performance achieved from correct use of search phases as seen in Chapter 4. Again we see, particularly for the larger instances, a dramatic improvement is obtained based purely on how the search phases are implemented.

Table 5.1: Comparison of the CP model with different search phases with 1 minute solve time. The symbol (*) indicates that the model was able to prove the instance to optimality, (-) indicates no feasible solution was found.

Inst.	Size	LB-T	LB-R	None	Low	High	Complete
1	(3-5-2)	299	244	**299***	**299***	**299***	**299***
2	(3-5-3)	394	210	**394***	**394***	**394***	**394***
3	(3-5-5)	441	386	**583**	**583**	**583**	**583**
4	(5-5-2)	693	502	**693***	**693***	**693***	**693***
5	(5-5-3)	665	392	**665***	**665***	**665***	**665***
7	(5-10-2)	1122	950	1159	1158	**1154**	1159
8	(5-10-3)	1076	1096	1271	1242	**1224**	**1224**
9	(5-10-5)	1150	984	1326	1334	**1306**	**1306**
10	(10-10-2)	2805	1517	**2805***	**2805***	**2805***	**2805***
11	(10-10-3)	2665	1725	2713	**2665***	2682	2680
12	(10-10-5)	2945	2109	3047	3046	2956	**2946**
13	(15-20-2)	10315	4762	10315*	16996	**10315***	**10315***
14	(15-20-3)	10685	6122	10952	14615	**10685***	**10685***
15	(15-20-5)	11248	6334	46729	20391	11522	**11375**
16	(20-10-2)	2132	5538	5980	6018	**5764**	5806
17	(20-10-3)	2186	4022	10520	5601	**5247**	5452
18	(20-10-5)	2127	5706	7560	8066	7359	**7276**
19	(30-20-2)	7308	12968	-	52993	20797	**20555**
20	(30-20-3)	7407	19070	-	37420	25788	**25425**
21	(30-20-5)	7650	25508	-	57674	**32688**	**32688**
22	(20-30-2)	23325	9329	-	-	-	**23325***
23	(20-30-3)	25050	11357	-	-	**25050***	**25050***
24	(20-30-5)	23049	13685	-	-	24397	**24261**

5.6 Conclusion

This chapter formalised the novel concept of driving variables as well as the novel local consistency rule LU consistency. We differentiated driving variables from

related concepts such as functional variable dependencies, w-cutsets to constraint networks, and backdoors to typical case complexity. We proved in a number of special cases of known global constraints that existing filtering algorithms and consistency checks are LU consistent for various partitions over the set of variables. We then showed how driving variables and LU consistency is used to significantly improve the performance of CP Optimizer through the correct application of search phases to the LHRSP+.

There are a number of directions for future work. Firstly, we would like to further understand the relationship between driving variables and backdoors to typical case complexity. In essence, driving variables are simply a relaxation of the definition of backdoors such that constraint propagation can be considered as part of the sub-solver, as strictly constraint propagation takes pseudo-polynomial time to reach a fixed point or infer failure. However, in practice it is often possible to define the constraints to a problem such that constraint propagation is performed quickly. Thus having more theoretical insight into understanding when constraint propagation will require worse-case complexity would be beneficial. Secondly, because the computational study is quite brief, we wish to evaluate this theory on other problems.

Thirdly, our concept of LU consistency seems to be related to unimodularity in linear programming (LP). Recent papers such as that by (Baatar et al., 2015) show how within a MIP framework it can be beneficial to branch upon certain decision variables such that the resulting problem becomes unimodular. This has many parallels to the work completed in this chapter. For the precedence constraints and the sequencing decisions with setup times these can be mapped easily to unimodular matrices in LP. However for the state and cumulative constraints considered this mapping is not immediately clear. Lastly, LU consistency can also be related with the notion of Necessary Truth Criterion in AI Planning & Scheduling, as utilised by Laborie (2003b). If a resource temporal network, which can be thought as a generalisation of state and cumulative functions, satisfies the NTC, then proving it is LU-consistent should be straight forward, as the temporal network is satisfied.[5] Formalising this connection seems like a worthwhile exercise.

In the next chapter we build upon the concepts of driving variables and LU consistency to create a framework to help reason about scheduling problems at different levels of abstractions by proposing the concept of interval clusters. We will consider the real-world problem in complete detail.

[5]Thank you to Dr Philippe Laborie for making this observation.

CHAPTER 6

Interval Clusters

6.1 Introduction

The ability to reason at different levels of abstraction has been a fundamental part of modelling and solving varying types of practical combinatorial problems involving automated reasoning. The quintessential example from the field of Planning in Artificial Intelligence is the concept of Hierarchical Task Networks, where the dependency among actions can be given in the form of hierarchical structured network and different levels in the hierarchy may correspond to different orders of abstraction. Inspired by this concept, and building upon the notion of driving variables and LU consistency introduced in the previous chapter, in this chapter we introduce a framework for building models to Constraint-Based Scheduling systems that explicitly consider multiple levels of abstraction and can still be solved efficiently.

More explicitly, we model the problem such that is possible to partition the set of all variables into *interval clusters*, which represent subsets of variables containing exactly one driving variable as well as satisfying a number of sufficient conditions involving connectivity and boolean logic. In doing so, we structure the problem in such a way that each cluster can be thought of as a single higher level variable and the sufficient conditions ensure desirable behaviour over the remaining variables. We demonstrate the benefit of considering interval clusters when modelling an

industrial scheduling of a fully automated advanced cell staining instrument.

In the previous chapter we saw how by considering driving variables and LU consistency it is possible to partition the set of variables into those driving variables that should be used as choice-points, and the auxiliary variables that are simply there to define constraints in a natural way. Here we define interval clusters as follows.

Definition 6.1.1. An *interval cluster* $V \subseteq \mathcal{V}$ is a subset of the interval variables such that

- (Unique) Each cluster contains a single driving variable x.
- (Connected) The start and end nodes of each interval variable in a cluster are in the same strongly connected component in the temporal network.
- (Optionality) The presence status of all auxiliary variables are equal to the presence status of the driving variable from the cluster, i.e., the auxiliary variables are present if and only if the driving variable is present.

Interval clusters provide a framework for decomposing a complicated scheduling problem into (1) a top level description of the problem in terms of driving variables, and (2) cluster level descriptions for all of the different possible interval clusters that must be considered. To demonstrate this concept, consider the temporal network in Figure 6.1, where nodes represent interval variables and arcs represent generalised precedences. The driving variables are coloured different colours depending on the structure of their interval clusters and the auxiliary variables are grey. On the left the full precedence graph is given. Although this representation is valid, it does not take into account any of the structure of the problem. Instead, on the right we demonstrate how the problem can be represented by a top level description of the driving variables (top right), and then descriptions for the various interval clusters (bottom right). Note that at the cluster level description, in the temporal network each cluster is a strongly connected network.

In this chapter we consider the real-world scheduling problem of how the next-generation of Leica Biosystem's Advanced Cell Staining Instrument processes multiple slides in parallel according to strict staining protocols in order to maximise the throughput of the machine. In the next section we will first provide a detailed description of a protocol, then provide a detailed description of the different components of the automated system, then discuss the objective of the scheduling problem.

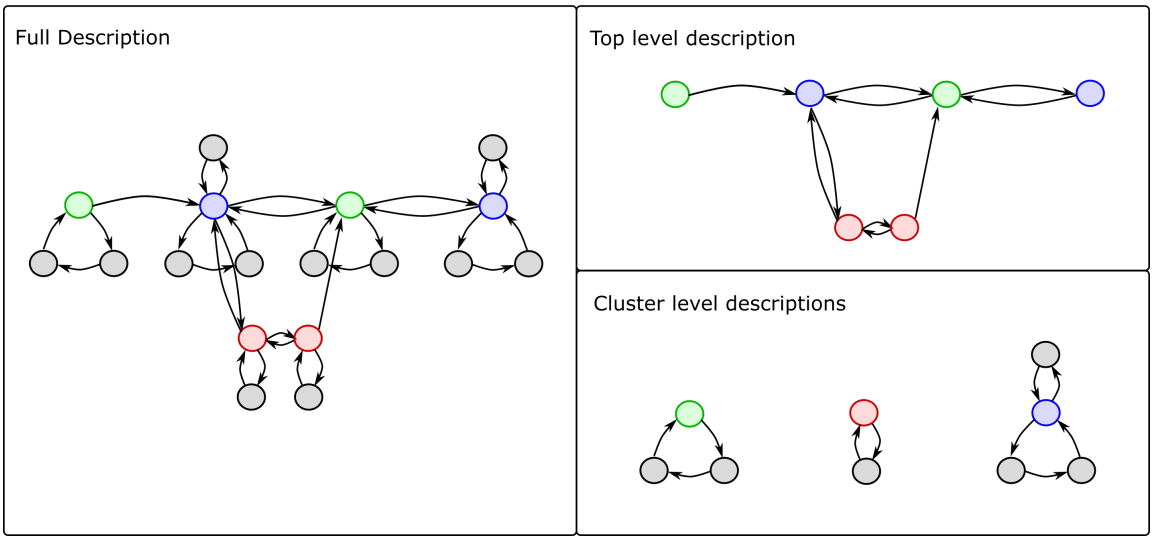


Figure 6.1: A visualisation that demonstrates how a full temporal network of a problem can be represented through a top level description of the graph induced by the driving variables and individual descriptions of the different types of clusters.

6.2 Problem Description

6.2.1 Protocols

Protocols are similar to cooking recipes, they provide a systematic set of instructions that if followed correctly will achieve desirable chemical reactions. There are two different types of protocols; (1) *staining protocols*, (2) *mixing protocols*. [1] A staining protocol refers to the set of steps that must be completed in order to process a slide correctly. A mixing protocol refers to the set of steps that must be completed in order to create a certain quantity of special types of chemicals known as *mixed chemicals*. Mixed chemicals have a short lifespan and thus must be created from its components parts within a certain amount of time before they can be applied to the samples. Although the outcomes are different, these different types of protocols are defined with respect to the same components.

A protocol consists of an ordered set of *steps*. Examples of a staining protocol and a mixing protocol are provided in Table 6.1. A protocol step consists of the following;

a) *Step Number (s)* - the ordinal number of the protocol stage.

b) *Main Action (a_s)* - the *main action* of a step may correspond to a "Dispense",

[1] In reality there are also cleaning protocols, which have been omitted in this chapter for simplicity.

a "Mix", or "None". A "Dispense" represents either a chemical being placed in contact with the sample for a staining protocol or a chemical being added to a mixing vial for a mixing protocol. A "Mix" is only relevant to mixing protocols and represents the situation where no additional chemical is added to the mixing vial but instead the chemicals already in the vials are mixed together to encourage a chemical reaction. The option "None" refers to a step that neither adds chemical or combines chemical but some other step parameters are required. This option provides more flexibility when defining protocols.

c) *Chemical (c_s)* - the chemical associated with the step. If the action is "Dispense" then the chemical is dispensed either onto the slide or into a mixing vial during that step. If the action is "Mix" then this corresponds to the chemical that is produced based on the action of mixing.

d) *Minimum Time (δ_s^{min})* - the minimum amount of time that must elapse before the following step can commence. This allows a sufficient time to allow the required chemical reactions to occur.

e) *Maximum Time (δ_s^{max})* - the maximum amount of time that can elapse before the following step must commence. If some chemical reactions occur for too long then the quality of the stain can be severely compromised and even the sample itself may be damaged.

f) *Scavenge* (β_s^{scav}) - A *scavenge* refers to removing a previously applied chemical from the sample prior to applying the next chemical. It is important that the scavenge occurs immediately before applying the next chemical to prevent the sample from being exposed to air and drying out. In general a scavenge is required for all but the first of the dispensing steps however occasionally there are situations where it is not required.

g) *Agitations* (n_s^{ag}) - An *agitation* refers to the process of moving chemical back and forth at a controlled rate over the sample and as such is only relevant to staining protocols. Typically when a chemical is dispensed onto a slide the chemical remains static. However for some steps it may be necessary to agitate the chemical to better facilitate the reaction between the chemical and the sample. If the number of agitations n_s^{ag} in a protocol step is greater than 0, then these agitations must take place within certain intervals of one another, $[\overline{\delta}_s^{ag}, \underline{\delta}_s^{ag}]$. These intervals start from after the second temperature ramp, if one is required, otherwise after the main action until the minimum time of the step has been reached. With respect to Step 2 of the staining protocol in

Table 6.1, an agitate must occur every 50-60 seconds for 120 seconds after the second temperature change is been completed.

h) *Microscavenge* (n_s^{micro}) - A *microscavenge* refers to the process of removing trapped bubbles from a sample by removing excess air from the slide. Microscavenging is only relevant to staining protocols. Similar to agitations, if microscavenging is required then a number of microscavenges, n_s^{micro}, take place within certain intervals, $[\bar{\delta}_s^{micro}, \underline{\delta}_s^{micro}]$. These intervals are taken from the end of the second temperature ramp, if one is required, otherwise from the end of the main action. With respect to Step 3 of the staining protocol in Table 6.1, a microscavenge must occur every 180-200 seconds for 600 seconds after the main action of the step starts.

i) *First Temperature Change* (Δ_s^1) - A *first temperature change* is used to indicate the increase or decrease in temperature which ends immediately before the scavenge if required or otherwise immediately before the main action of the step.

j) *Second Temperature Change* (Δ_s^2) - A *second temperature change* is used to indicate the increase or decrease in temperature required immediately after the completion of the main action.

Table 6.1: Example of a staining protocol (top) and mixing protocol (bottom)

	s	a_s	c_s	δ_s^{min}	δ_s^{max}	β_s^{scav}	$[\bar{\delta}_s^{ag}, \underline{\delta}_s^{ag}]$	$[\bar{\delta}_s^{micro}, \underline{\delta}_s^{micro}]$	Δ_s^1	Δ_s^2
Staining	1	Dispense	Red	30	30	×	×	×	0	0
	2	Dispense	Blue	120	130	✓	[50,60]	×	0	10
	3	None	-	600	800	×	×	[180,200]	-10	0
	4	Dispense	Green	60	120	✓	×	×	0	0
Mixing	1	Dispense	Blue	30	60	×	×	×	×	×
	2	Dispense	Yellow	120	150	×	×	×	×	×
	3	Mix	GreenMix	90	300	×	×	×	×	×

Figure 6.2 helps visually summarise the relative timings of a protocol step associated with a "Dispense" or a "None". The protocol step begins with the first temperature change. Immediately following this is the scavenge step if required, which removes the previous chemical from the sample. The main action begins at the end of the scavenge followed again immediately by a second temperature change if required. After the desired temperature has been reached the sample is maintained at a constant temperature. Microscavenges and agitations occur at specified intervals after this point. The main action of the next step must begin

between the minimum and maximum times. A "Mixing" step is much simpler as it does not require scavenges, agitates, microscavenges, or temperature changes.

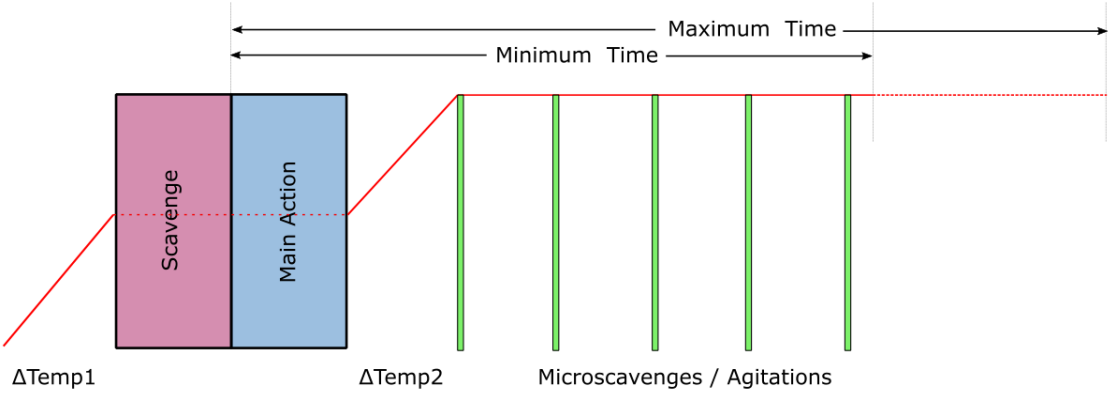


Figure 6.2: A visualisation of a possible protocol step

6.2.2 System Description

Two schematics of the system are provided and are referred to frequently during this overview of the system. The first schematic, Figure 6.3, provides a conceptual layout of the system. This layout will be used to explain the general work flow of the samples and the interaction that they have with the different components of the system. The second schematic, Figure 6.4, provides a conceptual layout of the internal mechanisms of the system. The internal view is useful in explaining how chemicals can be transferred through the system and also how vacuums are used for different purposes.

Before the samples are placed into the system laboratory technicians must perform some preparation. This preparation is not relevant to the scheduling problem being described however provides context to the workflow. A tissue sample is taken from a patient by biopsy and sent typically to a large pathology laboratory for testing. Before the sample can be stained and viewed it must be prepared so that a very thin section, typically 2-7μm, can be cut and placed onto a microscope slide. The slides are screened using an initial round of preliminary staining, known as Hematoxylin and Eosin staining (Mayer, 1891). This routine staining provides the pathologist with a very detailed view of tissue, which is often sufficient to allow a disease diagnosis. However if more information is required the pathologist will require the slide to undergo advanced cell staining according to a specific staining protocol. The laboratory technician places the slide into the *input drawer* of the system and then removes the from the *output drawer* after the slide is processed.

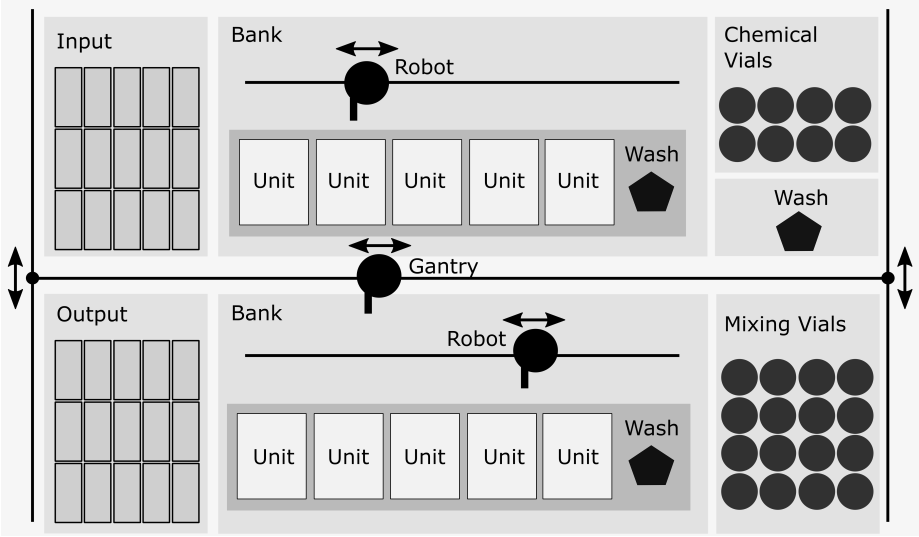


Figure 6.3: A conceptual layout of the automated cell staining system.

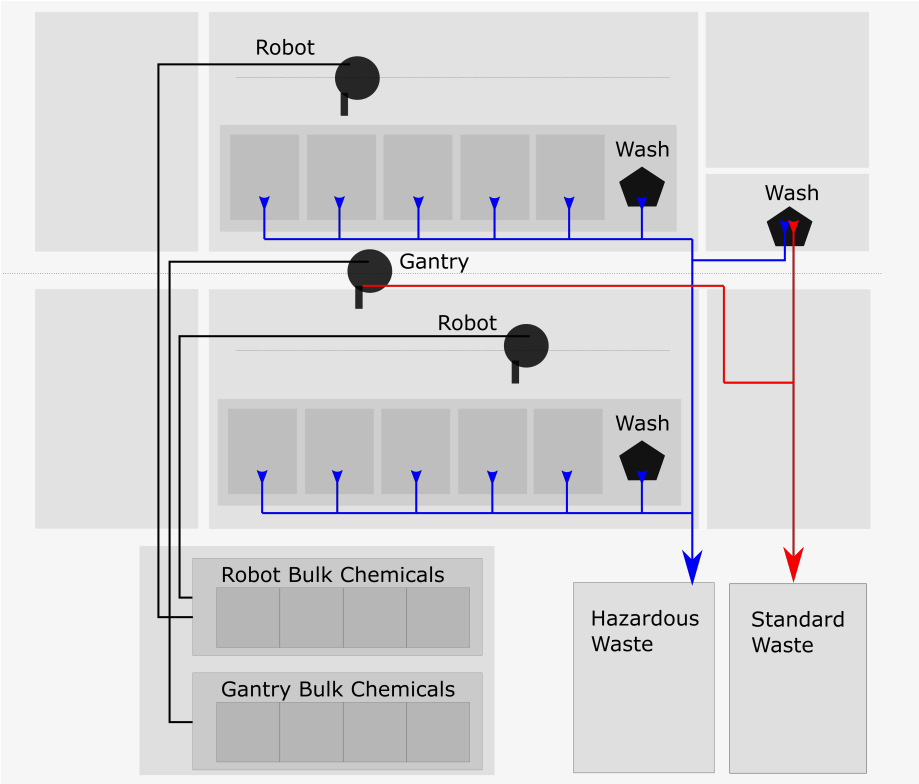


Figure 6.4: A conceptual layout of the internal mechanisms of the automated cell staining system.

A multi-purpose robot, referred to as the *gantry*, is able to transfer the slides one at a time from the input drawer to individual *processing units*, referred to from now on as simply *units*. The gantry transfers the slides using its *liquid handling probe*, or simply its *probe*. Units are designed to facilitate the correct interactions

between chemicals and the samples. The units are arranged across different *banks*. Once slides are in a unit they can then be processed by the system according to their staining protocol, a detailed description about how protocols are transformed into activities that the system can process is given in the following section. If the staining protocol requires a mixed chemical then the chemical must be mixed in a *mixing vial* according to its mixing protocol. Once a slide has finished processing the gantry can transfer the slide from its unit to the *output drawer* where the laboratory technician can receive it and forward it for examination.

The system has a number of different mechanisms for transferring chemical between locations. First, the three different methods of transferring chemical to the units and mixing vials will be explained followed by an explanation of how chemical is removed from both the units and the system.

The first method of transferring chemical to a slide is for so called *bulk chemicals*. A bulk chemical is a chemical that is frequently used across multiple protocols. As seen in Figure 6.4, bulk chemicals are directly connected to the gantry as well as to secondary types of robots referred to as *bank robots* or simply *robots*. There is a single robot per bank. In order to dispense a certain bulk chemical, either the appropriate robot (depending on the bank) or the gantry moves directly to the corresponding location and dispenses the chemical directly. The gantry does not require the use of its probe to dispense bulk chemical. A specific bulk chemical is either connected to the robots or the gantry but not both and thus there is never a choice of which resource should dispense which chemical.

The second method of transferring chemical is for *primary chemicals*. Primary chemicals are less frequently used than bulk chemicals. [2] Given that there are many different primary chemicals, it is not possible to directly connect all of them to the robots as done with the bulk chemicals. Instead primary chemicals are contained in *chemical vials*, seen in Figure 6.3. To transfer the chemicals from the chemical containers to a unit or a mixing vial the gantry uses its probe. Hence to transfer primary chemical, the gantry must (1) move to the appropriate chemical container, (2) aspirate the correct amount of chemical, (3) move to the correct unit, (4) dispense the chemical. Furthermore the probe has a finite capacity so there is a limit to the amount of chemical that it can carry at once and the probe must be washed before aspirating a different chemical to ensure no cross contamination occurs.

The third method of transferring chemical to a slide is for the *mixed chemicals*.

[2] Common examples of primary chemicals are all of the different antibodies used in immunohistochemistry, where each protocol might require a unique antibody.

Mixed chemicals are similar to primary chemicals in the sense that they must be transferred by the probe of the gantry. However they have the added complexity that they expire in a very short amount of time once being created. Thus mixed chemicals must be first produced by the system according to its mixing protocol. To do so the gantry must transfer components of the mixed chemical from chemical vials to the mixing vials, seen in Figure 6.3 and mix the chemicals according to the mixing protocol. We use the term *high-value* (HV) chemicals to refer to a chemical that is either a primary chemical or a mixed chemical.

Chemicals are removed from the system as waste. There are two different types of waste; (1) *hazardous waste* and (2) *standard waste*, as seen in Figure 6.4. The motivation behind this distinction is that laboratories must pay to dispose of hazardous waste, whereas it is safe to dispose of standard waste locally. Hence by separating the waste this helps laboratories avoid excessively large hazardous waste costs. The two different waste containers have individual vacuum systems, the *hazardous vacuum* and the *standard vacuum*, that are used to draw the waste chemical into the appropriate waste container. The vacuums must be held at different pressures depending on their function and it takes a certain amount of time for the vacuum to change pressure.

The hazardous vacuum is used to remove chemical to the hazardous waste container. It is connected to each unit and each of the *washing stations* — the gantry as well as each robot has its own *wash station*. Chemicals, due to their toxicity, are either considered *hazardous* or *safe*. If the primary or mixed chemical that was most recently in the probe is hazardous, then the hazardous vacuum is used when the probe is washed. Any chemical that has come into contact with the sample is considered hazardous. Therefore the hazardous vacuum is used to scavenge the chemicals from the units. The robots only require washing after agitating the sample, and as the robots have the potential to come into contact with the sample in this process, all of washes for the robots are considered hazardous. Finally the hazardous vacuum is also used to complete the appropriate microscavenges and in order to do so is typically on a low pressure setting.

The *standard vacuum* is primarily used to remove safe chemical to the standard container. As any liquid that has been in contact with a sample is considered hazardous, the only point where non-toxic liquids are removed from the system is at the wash station of the gantry. With respect to removing waste, the standard vacuum is only used when washing the probe after transferring a safe chemical. As this expected workload is considerably less than that of the hazardous vacuum, the

standard vacuum also has another purpose. The standard vacuum is connected to the probe of the gantry, as shown in Figure 6.4 and is used to transfer slides between the units and input/output drawers. As the pressure required to transfer slides is considerably more than required during the waste removal, the vacuum must be charged to the high pressure setting before the transfer of slides.

Finally the system has the capability to heat, cool and maintain the units at constant temperature. Each unit has its own heat pump. The heat pump can be used for both heating and cooling. It is assumed that these heat pumps can change the temperature of the unit at a constant rate regardless of whether it is heating or cooling. In order to operate, the heat pumps require a significant amount of energy. The system has a limited power budget, which restricts the number of heat pumps that can run in parallel. This must be taken into account when scheduling the usage of the heating pumps.

Once slides are in the input drawer a scheduling algorithm is run to coordinate the resources of the system such that the slides are loaded into the system, completed according to their specific staining protocols, and unloaded to the output drawer. Furthermore the system is designed such that laboratory technicians can place slides into unoccupied slots in the input drawers at any time, and the scheduling algorithm will adapt the existing schedule to include the new activities that must be completed. The system only becomes aware of the slides once they are placed into the input drawer. Hence, at least at first, the system must wait until a schedule is created. Clearly this is a dynamic, online scheduling problem.

6.2.3 Objectives

The overarching aim of the machine is to process all the slides according to their staining protocols as fast as possible without making any mistakes. All slides are considered equally important. Large pathology labs can run multiple systems in parallel and want as high a throughput as possible. Hence the objective of the scheduler is to quickly find a schedule for the slides in the input drawer, plus the slides currently being processed, such that all slides are processed in the least amount of time, i.e., minimise makespan. In practice it is generally accepted that it might take up to a minute to determine a reasonable schedule.

There are a number of secondary objectives that are of interest but are currently not considered explicitly. This includes: (a) minimising the amount of mixed chemical that does not get used as this increases cost of operation, (b) maximising the consistency between multiple slides of the same protocol, and (c) minimising

total completion times so that some slides can be removed as early as possible.

6.3 Model Description

In this section we demonstrate how interval clustering provides a framework for modelling this complicated industrial scheduling problem. First we introduce the require notation and parameters of an instance to the above scheduling problem. We will then describe how staining and mixing protocols are converted into project classes that give a top level description of the model. We then describe the different primitive actions that the automated system can complete and demonstrate how the top level description of the model can be translated into various interval clusters in terms of driving variables.

6.3.1 Notation

A summary of the notation used to represent the various sets and indices is provided in Table 6.2, and various parameters of the problem are provided in Table 6.3. It is worthwhile noting that we assume that the durations specified in Table 6.3 are independent of the actual amount of chemical that is being aspirated, dispensed, mixed, etc. In practice this is a reasonable assumption as the handling of chemicals actually makes up one portion of this time. For example, in the time allocated to dispense primary chemicals, d^{disp} the gantry must lower the pipette into the unit, dispense the chemical, and then lift the pipette back up to the height it is kept at in transit.

6.3.2 From Protocols to Project Classes

A protocol represents an ordered set of instructions. Each protocol is converted to a project class Q. Each class $q \in Q$ can be represented as a network where the nodes represent interval clusters $i \in V_q$ and the arcs represent the generalised precedence constraints between these clusters $(i, i') \in A_q$. The specifics of the different interval clusters will be defined in a later section, for now it is sufficient to think of the clusters as individual interval variables. As each cluster has a unique driving variable to an extent this is in fact a valid interpretation. Staining protocols and mixing protocols are converted to project classes by different methods.

Table 6.2: Summary of notation used for various sets

Index	Set	Description
q	Q	Project classes.[i]
	$Q^{staining} \subseteq Q$	Staining project classes.
	$Q^{mixing} \subset Q$	Mixing project classes.
(p,q)	P	Projects.
	$P^{critical} \subseteq P$	Critical projects that must start by $t^{critical}$.
p	$P_q \subseteq P$	Projects of class $q \in Q$.
(i,p,q)	V	Activity clusters.[ii]
	$V^{type} \subseteq V$	Activity clusters of a specific type.[iii]
(i,i')	A_q	Generalised precedence constraints in class $q \in Q$.[iv]
b	B	Banks.
u	U	Units.
	$U_b \subseteq U$	Units on bank $b \in B$.
c	C	Chemicals.[v]
	$C^{primary} \subseteq C$	Primary chemicals.
	$C^{mixed} \subseteq C$	Mixed chemicals.
	$C^{bulk} \subseteq C$	Bulk chemicals.
	$C^{gantry} \subseteq C^{bulk}$	Bulk chemicals connected to the gantry.
	$C^{robot} \subseteq C^{bulk}$	Bulk chemicals connected to the robots.
	$C^{safe} \subseteq C$	Chemicals considered safe.
	$C^{haz} \subseteq C$	Chemicals considered hazardous.
l	L	Locations.[vi]
	$L^{units} \subset L$	Locations of the units, i.e., $\bigcup_{u \in U} L_u^{unit}$.
	$L_b^{bank} \subseteq L^{units}$	Locations of the units on bank $b \in B$, i.e.,$\bigcup_{u \in U_b} L_u^{unit}$.
	$L^{gantry} \subset L$	Locations that the gantry can access.
	$L_b^{robot} \subset L$	Locations that the robot on bank $b \in B$ can access.
ω	Ω	Pressure levels of vacuums.
	$\Omega^{haz} \subseteq \Omega$	Pressure levels require by the hazardous vacuum.
	$\Omega^{standard} \subseteq \Omega$	Pressure levels require by the standard vacuum.

[i] Each staining and mixing protocol has a corresponding project class as defined in Section 6.3.2.
[ii] Activity clusters are groups of activities associated with the same index.
[iii] The different cluster types are summarised in Table 6.4, so $V^{Dispense}$ would correspond to the cluster associated with the dispense of a primary chemical.
[iv] As generalised precedence relations are only defined between activity clusters from the same class we do not introduce a set of of these relations.
[v] The set of chemicals can either be classified into the three disjoint subsets $C^{primary}$, C^{mixed}, and C^{bulk} based on the chemical type, or into the two disjoint subsets C^{safe} and C^{haz} based on the toxicity of the chemical.
[vi] The set L consist of the location of the input drawer L^{input}, output drawer L^{input}, chemical vials L^{vials}, mixing vials L^{mixing}, probe wash station L^{wash}, robot wash station L_b^{wash} on each bank $b \in B$, and the units L^{units}.

Table 6.3: Summary of model parameters

Notation	Description
$\delta_{i,i',q}$	Timelag associated with arc $(i, i') \in A_q$ of class $q \in Q$
$\Delta^1_{i,q}$	First temperature change associated with cluster $i \in V_q$ of class $q \in Q$
$\Delta^2_{i,q}$	Second temperature change associated with cluster $i \in V_q$ of class $q \in Q$
$\beta^{scav}_{i,q}$	Binary parameter equal to 1 if cluster $i \in V_q$ of class $q \in Q$ requires a scavenge, 0 otherwise
$t^{critical}$	The time before which all critical projects must start by.
$d^{aspirate}$	The time to aspirate chemical by the pipette
d^{disp}	The time to dispense chemical by the pipette
d^{bulk1}	The time to dispense bulk chemical by the gantry
d^{bulk2}	The time to dispense bulk chemical by a bank robot
d^{clean}	The time to clean a mixing vial
d^{mix}	The time to mix the contents of a mixing vials
d^{haz}	The time to wash a hazardous chemical from the pipette
$d^{standard}$	The time to wash a safe chemical from the pipette
d^{wash}	The time to wash the bank robot
d^{scav}	The time to scavenge a unit
d^{micro}	The time to microscavenge a unit
d^{pickup}	The time to pickup a slide from the input drawer
d^{place}	The time to place a slide into the output drawer
d^{load}	The time to load a slide into a unit
d^{unload}	The time to unload a slide from a unit
$d^{agitate}$	The time to perform agitation
d^{temp}	The time required to change the temperature of a unit by 1 degree
$d^{gantry}_{l,l'}$	The travel time for the gantry to move between locations $l, l' \in L^{gantry}$
$d^{robot}_{b,l,l'}$	The travel time for the robot on bank $b \in B$ to move between locations $l, l' \in L^{robot}_b$
$d^{standard}_{\omega,\omega'}$	The recharge time of the standard vacuum between states $\omega, \omega' \in \Omega^{standard}$
$d^{haz}_{\omega,\omega'}$	The recharge time of the hazardous vacuum between states $\omega, \omega' \in \Omega^{haz}$
$c_{i,q}$	The chemical required by cluster $i \in V_q$ of class $q \in Q$
$c(q)$	The mixed chemical produced by project class $q \in Q^{mix}$
z_c	The number of units of mixed chemical $q(c) \in C^{mixed}$ produced by completing a project from class $q \in Q^{mix}$
R^{mix}	The number of mixing vials available
R^{power}	The number of temperature ramping activities that can occur concurrently
R^{probe}	The capacity of the probe

Staining Protocols

The steps of a staining protocol are converted into a project class with seven different types of interval clusters. Recall that intuitively an interval cluster represents a higher level description of a set of primitive tasks that must be completed. The seven different interval clusters required for the project classes from staining protocols are:

1. *Load* - transferring a slide from the input drawer to the required unit;
2. *Bulk* - dispensing a bulk chemical onto the slide;
3. *HV Dispense* - dispensing a HV chemical onto the slide;
4. *Agitate* - using a bank robot to move chemical back and forth across a sample to accelerate a chemical reaction;
5. *Micro* - removing bubbles from the sample through short bursts of the hazardous vacuum; and
6. *None* - an additional cluster reserved for steps of the protocol where no chemical is dispensed yet a scavenge is required or temperature needs to be changed.
7. *Unload* - transferring a slide from the unit to the output drawer.

Algorithm 3 provides a formal description of how a staining protocol is converted into a project class. Figure 6.5 demonstrates the project class created for the staining protocol specified in Table 6.1. All project classes begin with a Load cluster. Each step of the protocol is then converted into either a Bulk, HV Dispense, or None cluster, depending on what type of chemical (if any) is required in that step. If the step of the protocol requires agitation or microscavenging, then the number of Agitate or Microscavenge clusters required is determined based on the ratio of the minimum time required by the step. For example in step 2 of the example staining protocol given in Table 6.1 the minimum time of the step is 120 time units and the minimum time between agitates is 50 time units, thus two agitate clusters are required. Likewise, in step 3 of the example staining protocol the minimum time of the step is 600 time units and the minimum time between microscavenges is 180 time units, thus three microscavenge clusters are required. All project classes end with an unload cluster.

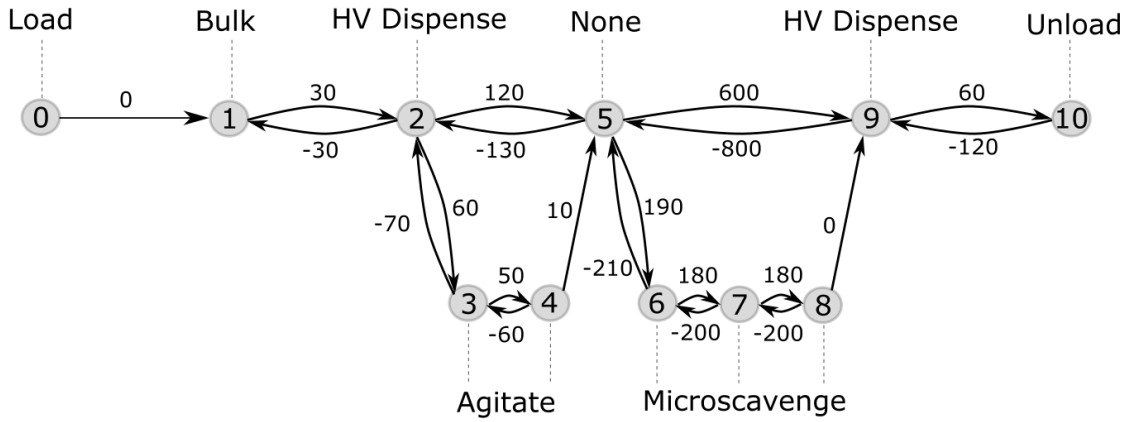


Figure 6.5: A staining project class.

Algorithm 3 Converting staining protocols to project classes.

1: Initialise the class with a load node $V_q = \{\texttt{Node}(0, load, -, 0, 0, 0)\}$, a single arc $A_q = \{\texttt{Arc}(0, 1, 0)\}$, and set $i = 1$
2: **for** each step s in protocol **do**
3: $type = \texttt{InferType}(c_s)$
4: $\texttt{addNode}(i, type, c_s, \Delta_s^1, \Delta_s^2, \beta_s^{scav})$
5: **if** $n_s^{ag} > 0$ **then**
6: Set $pred := i$, $succ := i + 1$
7: **for** each agitate **do**
8: $\texttt{addArc}(pred, succ, \delta_s^{ag})$, $\texttt{addArc}(succ, pred, -)\delta_s^{ag})$
9: $\texttt{addNode}(succ, agitate, c_s, -, -, -)$
10: $pred = succ$ and increment $succ$
11: $\texttt{addArc}(pred, i + 1 + n_s^{ag} + n_s^{micro}, 0)$
12: **if** $n_s^{micro} > 0$ **then**
13: Set $pred := i$, $succ := i + n^{ag} + 1$
14: **for** each microscavenge **do**
15: $\texttt{addArc}(pred, succ, \delta_s^{micro})$, $\texttt{addArc}(succ, pred, -\delta_s^{micro})$
16: $\texttt{addNode}(succ, micro, C_s, -, -, -)$
17: $pred = succ$ and increment $succ$
18: $\texttt{addArc}(pred, i + 1 + n_s^{ag} + n_s^{micro}, 0)$
19: $\texttt{addArc}(i, i + 1 + n_s^{ag} + n_s^{micro}, \delta_s^{min})$, $\texttt{addArc}(i + 1 + n_s^{ag} + n_s^{micro}, i, -\delta_s^{max})$
20: $i = i + 1 + n_s^{ag} + n_s^{micro}$
21: $\texttt{addNode}(i, unload, -, 0, 0, 0)$

Mixing Protocols

A mixing protocol specifies the procedure to create a certain number of units of a mixed chemical. Before a schedule is generated it is not clear how many times a

single mixing protocol must be completed, i.e., how many projects from a project class based on a mixing protocol are processed. Reducing the number of mixing projects reduces the amount of work needed to be done by the system. However, given the mixed chemicals expire in a short amount of time, completing fewer mixing projects may impose unnecessarily strict temporal constraints on the corresponding dispenses from the staining projects. We wish to model this aspect of the problem in such a way that the solver can determine the correct number of mixing projects itself.

The steps of a mixing protocol are converted into a project class with four different types of interval clusters.

1. *HV Dispense* - dispensing a HV chemical into a mixing vial;
2. *Mix* - mixing chemicals in a mixing vial to encourage a reaction;
3. *Clean* - cleaning a mixing vial after it has been used such that another mixed chemical can be prepared in it; and
4. *Dummy* - a dummy variable that represents the dispensing of the mixed chemical created by a project on a possible staining project.

An `isomorphism` constraint will be used to link the Dummy clusters for a certain mixed chemical with the HV clusters from the staining projects that require that mixed chemical. The mixing projects where no dummy clusters are used in the mapping of the `isomorphism` constraint correspond to mixing procedures that are never used for staining. Hence these projects can be removed from scheduling or the corresponding interval variables set to absent.

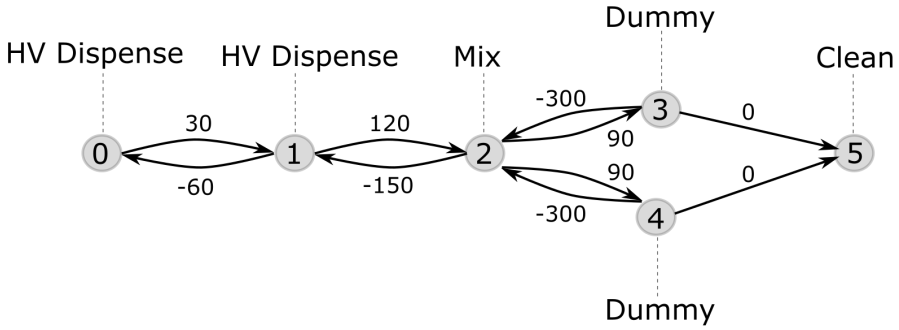


Figure 6.6: A mixing project class.

Algorithm 4 proves a formal description of how a mixing protocol is converted into a project class. Figure 6.6 visualises the project class created for the mixing protocol specified in Table 6.1. A node is created for each step in the protocol; if the main action of the step is to Mix then a Mix cluster is created, otherwise if the

main action is a Dispense then a HV Dispense cluster is created. After the last step a Dummy cluster node is created for each unit of mixed chemical produced, i.e., if a protocol creates 3 units of mixed chemical then 3 Dummy clusters are created. Finally, the project ends with a Clean cluster with simple precedences from each of the Dummy nodes.

For mixing project class, $q \in Q^{mix}$, the maximum number of instances of the class is equal to the number of HV Dispense clusters from the staining project classes requiring that mixed chemical, i.e., $|P_q| = |\{(i,p,q) \in V^{Dispense} : c_{i,q} = c(q)\}|$.

Algorithm 4 Converting mixing protocols to project classes

Initialization $V_q = \emptyset, A_q = \emptyset, i = 0$
for each step s **do**
    $type = \texttt{InferType}(c_s)$, $\texttt{addNode}(i, type, c_s)$
    if not the last step **then**
        $\texttt{addArc}(i, i+1, \delta_s^{min})$, $\texttt{addArc}(i+1, i, -\delta_s^{max})$
    increment i
store last step node $i' = i - 1$
store cleaning node i'' as i plus number of dummy nodes
for each dummy dispense required **do**
    $\texttt{addNode}(i, Dummy, c_s)$
    $\texttt{addArc}(i', i, \delta_s^{min})$, $\texttt{addArc}(i, i', -\delta_s^{max})$
    $\texttt{addArc}(i, i'', 0)$
    if not last dummy node **then**
        $\texttt{addArc}(i, i+1, 0)$
    increment i
$\texttt{addNode}(i, Clean, -, 0, 0, 0)$

6.3.3 Assigning Staining Projects to Units

The CP Optimizer model presented in this chapter assumes that each staining project has been assigned to a specific unit. This is a significant assumption and one that we intend to revisit in the future. Currently this assignment is completed using the following simple heuristic. Each staining class is assigned a *science time*, which is the sum of minimum times, δ_s^{min}, of each step s in the associated staining protocol. Each unit has a *vacant-time* associated with it that represents when it will possibly become vacant, which is originally set to zero. Project classes are then selected in non-increasing order of their respective class's science time. A project class $q \in Q^{staining}$ is selected and then all projects of that class $p \in P_q$ are iteratively

assigned to the unit with the lowest vacant-time, splitting ties based on the index of the unit. We denote the location of the projects as $l(p,q) \in L$ and the bank of the project $b(p,q) \in B$. When a project is assigned to a unit, the respective vacant-time is increased by the project's science time. It is important to note that, in this heuristic the choice of whether to assign projects of the same class to units on the same bank largely depends on the indices of the units. The location of all mixing projects are set to the L^{mixing}.

6.4 CP Optimizer Model

We will describe the model of this industrial scheduling problem by first providing a top level description of the problem where interval clusters will be represented by their unique driving variables. The different sequence variables, state functions, and cumulative functions used to represent different resources of the system will be described at this top level description. Models for each interval cluster will then be introduced in isolation. This will include the definition of the driving and auxiliary interval variables for each cluster. No sequence variables, state functions or cumulative functions are introduced at the cluster level description of the models. Finally, we will prove that any assignment over the driving variables that reaches a fixed point results in all remaining constraints becoming LU consistent for a specific partition of the auxiliary variables that are not functionally dependent on the driving variables.

6.4.1 Top Level Description

The top level description considers two types of interval variables. For each cluster $(i,p,c) \in V$ a driving variable $\alpha_{i,p,c}^{driving}$ is assumed to exist. The details of these driving variables, such as lengths and presence status, are defined at the cluster level. For each project $(p,q) \in P$, a project interval variable $\alpha_{p,q}^{project}$ is defined to represent the span of the individual project.

The different resources and components of the automated system are represented by a range of sequence variables, state functions and cumulative functions. Firstly, we will consider the gantry. The location of the gantry is represented by both a sequence variable ρ^{gantry} and a state function ψ^{gantry}. The gantry state function will consider all intervals where the gantry is constrained to be at a specific location, whereas the gantry sequence variable will only consider the subset of

these intervals that cannot overlap.[3] The types of the intervals in the gantry state function, as well as the values of the state function, represent the location of the gantry. A transition matrix representing the travel time required by the gantry between the different locations, $M^{gantry} = ((d^{gantry}_{l,l'})_{l \in L^{gantry}})_{l' \in L^{gantry}}$, will be used between different types of the gantry sequence variable and to transition between state values of the gantry state function. A state function, ψ^{carry}, is used to indicate the type of chemical the gantry is currently carrying in its pipette. Similarly, a state function, ψ^{contam}, is used to indicate which chemical, if any, is contaminating the pipette. The values of the carry and contaminate state functions indicate the type of chemical being carried or contaminating the pipette, respectively. A cumulative function, $f^{pipette}$, represents the amount of chemical currently in the pipette. A state function, ψ^{mixing}, is used to indicate which mixing project was used for each of the dummy intervals. This state function is used to ensure that different mixed chemicals are not aspirated from different mixing vials.

The pressure level of the hazardous and standard vacuums are each represented by a state function, ψ^{haz} and $\psi^{standard}$, respectively, and a sequence variable, ρ^{haz} and $\rho^{standard}$, respectively. Similar to how the gantry location is modelled, the state functions consider all intervals where the individual vacuums must be maintained at a certain pressure level, and the sequence variables only consider the subset of these intervals that cannot overlap. Transition matrices representing the recharge time between different pressure states are defined for both the hazardous vacuum, $M^{haz} = ((d^{haz}_{\omega,\omega'})_{\omega \in \Omega})_{\omega' \in \Omega}$ and standard vacuum, $M^{standard} = ((d^{standard}_{\omega,\omega'})_{\omega \in \Omega})_{\omega' \in \Omega}$. These transition matrices are considered by the state functions between different states and the sequence variables between different interval types.

The locations of the bank robots on each bank $b \in B$ are represented by individual sequence variables, ρ^{robot}_b. A state function is not required for the bank robots because, as will be seen at the interval level, all intervals requiring the bank robot cannot overlap. A transition matrix is introduced for each robot $b \in B$, to represent the travel time between the different locations for the robot on each bank $M^{robot}_{b,l,l'} = ((d^{robot}_{b,l,l'})_{l \in L^{robot}_b})_{l' \in L^{robot}_b}$.

The remaining resources of the system are modelled as follows. The amount of power being used by the system is represented by a cumulative function, f^{power}.

[3] Similar to the previous two chapters, state functions will be used to enforce location constraints in such a way that multiple intervals are allowed to overlap so long as they represent the same process. For example, in previous chapters the aspirate intervals represented the probe aspirating chemical from a single location.

The number of available clean mixing vials of the system is represented by a cumulative function, f^{mixing}. Each unit $u \in U$ is represented by a sequence variable ρ_u^{unit} that is defined without a transition matrix.

The top level description of the model can now be expressed as follows.

$$\min. \quad \max_{(p,q)\in P^{primary}} (endOf(\alpha_{p,q}^{project})) \tag{6.1a}$$

s.t. $(6.2a) - (6.8g)$

$$\texttt{startAtStart}(\alpha_{p,q}^{project}, \alpha_{1,p,q}^{drive}, \delta_{i,i',q}), \quad \forall q \in Q; p \in P_q \tag{6.1b}$$

$$\texttt{endAtEnd}(\alpha_{p,q}^{project}, \alpha_{|P_q|,p,q}^{drive}), \quad \forall q \in Q; p \in P_q \tag{6.1c}$$

$$\texttt{startBefore}(\alpha_{p,q}^{project}, t^{critical}) \quad \forall (p,q) \in P^{critical} \tag{6.1d}$$

$$\texttt{startBeforeStart}(\alpha_{i,p,q}^{drive}, \alpha_{i',p,t}^{drive}), \quad \forall (i,i') \in A_t, p \in P_q; q \in Q \tag{6.1e}$$

$$\texttt{isomorphism}(V_c^1, V_c^2), \quad \forall c \in C^{mix} \tag{6.1f}$$

$$\texttt{presenceOf}(\alpha_{p,q}^{project}) = \texttt{presenceOf}(\alpha_{i,p,q}^{drive}) \quad \forall (i,p,q) \in V \setminus \overline{V}^{Dummy} : q \in Q^{mix} \tag{6.1g}$$

$$\texttt{presenceOf}(\alpha_{p,q}^{project}) \geq \texttt{presenceOf}(\alpha_{i,p,q}^{drive}) \quad \forall (i,p,q) \in \overline{V}^{Dummy} \tag{6.1h}$$

$$\texttt{presenceOf}(\alpha_{i,p,q}^{drive}) \geq \texttt{presenceOf}(\alpha_{i+1,p,t}^{drive}) \quad \forall (i,p,q), (i+1,p,q) \in \overline{V}^{Dummy} \tag{6.1i}$$

$$\texttt{inSequence}(\rho_{u(p,q)}^{unit}, \alpha_{p,r}^{project}) \quad \forall (p,q) \in P^{mix} \tag{6.1j}$$

$$\texttt{noOverlap}(\rho^{gantry}, M^{gantry}), \tag{6.1k}$$

$$\texttt{noOverlap}(\rho^{haz}, M^{haz}), \tag{6.1l}$$

$$\texttt{noOverlap}(\rho^{standard}, M^{standard}), \tag{6.1m}$$

$$\texttt{noOverlap}(\rho_b^{robot}, M_b^{robot}), \quad \forall b \in B \tag{6.1n}$$

$$\texttt{noOverlap}(\rho_u^{unit}), \quad \forall u \in U \tag{6.1o}$$

$$f^{mix} \mathrel{+}= \texttt{pulse}(\alpha_{p,q}^{project}, 1) \quad \forall (p,q) \in P^{mix} \tag{6.1p}$$

$$f^{mix} \leq R^{mix}, \tag{6.1q}$$

$$f^{power} \leq R^{power}, \tag{6.1r}$$

$$f^{probe} \leq R^{probe}, \tag{6.1s}$$

$$\alpha_{p,q}^{project} \in \texttt{interval}(q \in Q^{prime}, (0,\infty)) \quad \forall q \in Q; p \in P_q \tag{6.1t}$$

$$f^{probe}, f^{power}, f^{mixing} \in \texttt{cumul} \tag{6.1u}$$

$$\rho^{standard}, \rho^{haz}, \rho^{gantry} \in \texttt{sequence} \tag{6.1v}$$

$$\rho_b^{robot} \in \texttt{sequence} \quad \forall b \in B \tag{6.1w}$$

$$\rho_u^{unit} \in \texttt{sequence} \quad \forall u \in U \tag{6.1x}$$

$$\psi^{gantry} \in \texttt{state}(M^{gantry}) \tag{6.1y}$$

$$\psi^{standard} \in \texttt{state}(M^{standard}) \tag{6.1z}$$

$$\psi^{haz} \in \texttt{state}(M^{haz}) \tag{6.1aa}$$

$$\psi^{carry}, \psi^{contam}, \psi^{mixing} \in \texttt{state} \tag{6.1ab}$$

The objective (6.1a) is to minimise the maximum completion time of any of the staining projects. Constraints (6.2a)-(6.8g) consider the constraints and interval variable definitions for the individual cluster, which will be presented in the following sections. Constraints (6.1b) and (6.1c) ensure the project intervals start at the start of the first driving interval of the project, and end at the end of the last driving interval of the project, respectively. Constraints (6.1d) ensure the critical projects start within the specified amount of time. Constraints (6.1e) ensure all of the precedences are respected between the driving variables of the different clusters in the same project. Constraints (6.1f) ensure the HV dispense clusters requiring mixed chemical are mapped to the dummy clusters appropriately, where $V_c^1 = \bigcup_{(i,p,q)\in V_c^{Dummy}} \alpha_{i,p,q}^{disp}$ and $V_c^2 = \bigcup_{(i,p,q)\in V_c^{Dispense}} \alpha_{i,p,q}^{disp}$. Constraints (6.1g) relate the presence status of the mixing project intervals to all the driving variables excluding all but the first Dummy Cluster from each project, i.e., $\overline{V}^{Dummy} = \{(i,p,q) \in V^{Dummy} : \exists i' < is.t.(i,p,q) \in V^{dummy}\}$. Constraints (6.1h) state that if a Dummy Cluster is completed then the whole mixing project must be present. Constraints (6.1i) force an ordering on the driving variables from Dummy Clusters.

Constraints (6.1j) state that the project intervals are considered by the appropriate unit sequence variable. Constraints (6.1k)-(6.1o) state that intervals in the gantry sequence, hazardous vacuum sequence, standard vacuum sequence, bank robot sequences, and unit sequences must not overlap and if appropriate respecting specific transition matrices, respectively. Constraints (6.1p) state that each mixing projects requires a single mixing vial by ensuring that the value of the mixing vial cumulative function increases by a single unit for the duration of the project. Constraints (6.1q)-(6.1s) ensure the mixing vial cumulative function, power budget cumulative function, and probe capacity cumulative functions never exceed the number of vials, the power budget, and the capacity of the probe, respectively. Constraints (6.1t) define the project interval variables. Finally, constraints (6.1u)-(6.1ab) define the various sequence variables, state functions, and cumulative functions that will also be used interval cluster.

6.4.2 Clusters Level Descriptions

The model consists of ten different types of interval clusters. A summary of the different interval clusters is given in Table 6.4. The start and end times of the interval variables from a cluster can be partitioned with respect to a set of datums.

All start and end times associated with a single datum are functionally dependent on one another. As the driving variable in each cluster can only be associated with a single datum, special attention is paid to clusters with more than one datum. We partition the datums that are not associated with a driving variable into a lower set $\mathcal{L}$ and an upper set $\mathcal{U}$. Once all the driving variables have been fixed and constraint propagation reaches a fixed point we assign all variables in $\mathcal{L}$ to their lower bound and all variables in $\mathcal{U}$ to their upper bound.

To reduce the amount of repetition, where possible, interval clusters with a lot of similarity will be introduced together.

Table 6.4: Variable clusters

Cluster Name	Driving Variable	Set Notation	Primary Projects	Mixing Projects	Nb. Intervals	Nb. Datums
Bulk	α^{disp}	V^{Bulk}	✓	×	4	1
None	α^{none}	V^{None}	✓	×	4	1
HV Dispense	α^{disp}	$V^{Dispense}$	✓	✓	8	4
Load	α^{lift}	V^{Load}	✓	×	3	1
Unload	α^{place}	V^{Unload}	✓	×	4	1
Agitate	$\alpha^{agitate}$	$V^{Agitate}$	✓	×	2	1
Microscavange	α^{micro}	V^{Micro}	✓	×	1	1
Mix	α^{mix}	V^{Mix}	×	✓	3	2
Clean	α^{clean}	V^{Clean}	×	✓	3	2
Dummy	α^{disp}	V^{Dummy}	×	✓	2	3

Bulk / None Clusters

A Bulk cluster, $(i, p, c) \in V^{Bulk}$, represents a step in the staining protocol where a bulk chemical $c \in C^{bulk}$ is dispensed on a slide either by the gantry or the relevant bank robot. Similarly, a None cluster, $(i, p, c) \in V^{None}$, represents a step in a staining protocol without a main action. Recall that steps without a main action are useful in practice to describe more complicated heating/cooling procedures or to scavenge not immediately before a dispense.

Both clusters consider four different types of interval variables. A dispense variable, $\alpha^{disp}_{i,p,q}$, represents when chemical $c \in C^{bulk}$ is being dispensed on the slide. For None Clusters the length of the dispense variable is set to zero. If $c \in C^{gantry}$

then the chemical is dispensed by the gantry and the interval has duration d^{bulk1}, otherwise if $c \in C^{bulk}$ then the chemical is dispensed by the bank robot and the interval has duration d^{bulk2}. Two interval variables, $\alpha^{temp1}_{i,p,q}$ and $\alpha^{temp2}_{i,p,q}$, represent the time to change the temperature (if any) of the relevant unit before and after the dispense, respectively. The duration of these temperature change intervals is equal to the required temperature change, $\Delta^1_{i,q}$ and $\Delta^2_{i,q}$, multiplied by the rate the units can change temperature, d^{ramp}. Where required, a scavenge interval $\alpha^{scav}_{i,p,q}$ is considered with fixed duration d^{scav} to represent the removal of the previously applied chemical. The dispense interval variable is considered the driving variable. The cluster level description of the Bulk and None Clusters can now be modelled as follows.

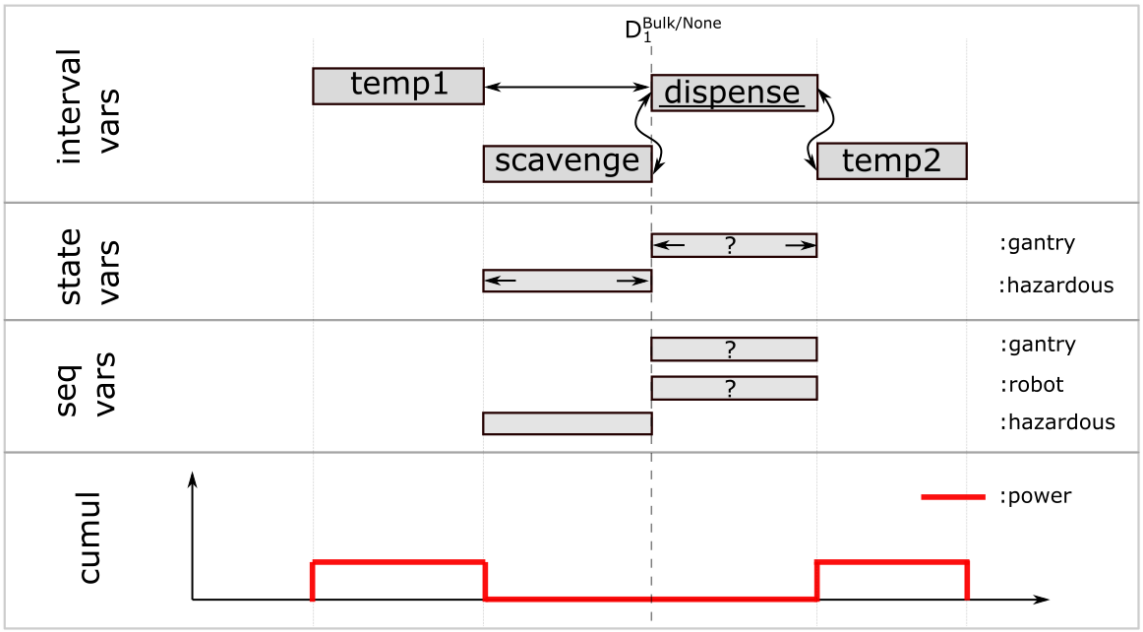


Figure 6.7: A visualisation of the Bulk Dispense cluster of interval variables and their interaction with the relevant sequence variables, state variables and cumul functions. Here the dispense variable is considered the driving variable. Note that either the gantry or the appropriate bank robot completes the dispense depending on the type of bulk chemical required - this is indicated by the "?" symbols.

$$\texttt{endAtStart}(\alpha^{temp1}_{i,p,q}, \alpha^{disp}_{i,p,q}, d^{scav}_{i,q} \cdot \beta^{scav}_{i,q}), \qquad \forall (i,p,q) \in V^{Bulk} \cup V^{None} : \Delta^{temp1}_{i,q} > 0 \tag{6.2a}$$

$$\texttt{endAtStart}(\alpha^{scav}_{i,p,q}, \alpha^{disp}_{i,p,q}), \qquad \forall (i,p,q) \in V^{Bulk} \cup V^{None} : \beta^{scav}_{i,p,q} \tag{6.2b}$$

$$\texttt{endAtStart}(\alpha^{disp}_{i,p,q}, \alpha^{temp2}_{i,p,q}), \qquad \forall (i,p,q) \in V^{Bulk} \cup V^{None} : \Delta^{temp2}_{i,q} > 0 \tag{6.2c}$$

$$\texttt{alwaysEqual}(\psi^{gantry}, \alpha^{disp}_{i,p,q}, l(p,q), \leftrightarrow) \qquad \forall (i,p,q) \in V^{Bulk} : c_{i,q} \in C^{gantry} \tag{6.2d}$$

$$\texttt{alwaysEqual}(\psi^{haz}, \alpha^{scav}_{i,p,q}, \omega^{med}, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Bulk} \cup V^{None} : \beta^{scav}_{i,q} \tag{6.2e}$$

$$\texttt{inSequence}(\rho^{gantry}, \alpha^{disp}_{i,p,q}, l(p,q)) \qquad \forall (i,p,q) \in V^{Bulk} : c_{i,q} \in C^{gantry} \tag{6.2f}$$

$$\texttt{inSequence}(\rho^{robot}_{b(p,t)}, \alpha^{disp}_{i,p,q}, l(p,q)) \qquad \forall (i,p,q) \in V^{Bulk} : c_{i,q} \in C^{robot} \tag{6.2g}$$

$$\texttt{inSequence}(\rho^{haz}, \alpha^{scav}_{i,p,q}, \omega^{med}) \qquad \forall (i,p,q) \in V^{Bulk} \cup V^{None} : \beta^{scav}_{i,q} \tag{6.2h}$$

$$f^{power} \mathrel{+}= \texttt{pulse}(\alpha^{temp1}_{i,p,q}, 1) \qquad \forall (i,p,q) \in V^{Bulk} \cup V^{None} : \Delta^{temp1}_{i,q} > 0 \tag{6.2i}$$

$$f^{power} \mathrel{+}= \texttt{pulse}(\alpha^{temp2}_{i,p,q}, 1) \qquad \forall (i,p,q) \in V^{Bulk} \cup V^{None} : \Delta^{temp2}_{i,q} > 0 \tag{6.2j}$$

$$\alpha^{driving}_{i,p,q} = \alpha^{disp}_{i,p,q} \in \texttt{interval}(comp, d^{bulk1}) \qquad \forall (i,p,q) \in V^{Bulk} : c_{i,q} \in C^{gantry} \tag{6.2k}$$

$$\alpha^{driving}_{i,p,q} = \alpha^{disp}_{i,p,q} \in \texttt{interval}(comp, d^{bulk2}) \qquad \forall (i,p,q) \in V^{Bulk} : c_{i,q} \in C^{robot} \tag{6.2l}$$

$$\alpha^{driving}_{i,p,q} = \alpha^{disp}_{i,p,q} \in \texttt{interval}(comp, 0) \qquad \forall (i,p,q) \in V^{None} \tag{6.2m}$$

$$\alpha^{temp1}_{i,p,q} \in \texttt{interval}(comp, d^{temp} \cdot \Delta^{1}_{i,q}) \qquad \forall (i,p,q) \in V^{Bulk} \cup V^{None} : \Delta^{1}_{i,q} > 0 \tag{6.2n}$$

$$\alpha^{temp2}_{i,p,q} \in \texttt{interval}(comp, d^{temp} \cdot \Delta^{2}_{i,q}) \qquad \forall (i,p,q) \in V^{Bulk} \cup V^{None} : \Delta^{1}_{i,q} > 0 \tag{6.2o}$$

$$\alpha^{scav}_{i,p,q} \in \texttt{interval}(comp, d^{scav}) \qquad \forall (i,p,q) \in V^{Bulk} \cup V^{None} : \beta^{scav}_{i,q} \tag{6.2p}$$

Constraints (6.2a) ensure that the first temperature change variables either end at the start of the corresponding scavenge variable, or if no scavenge is required at the start of the corresponding dispense variable. Constraints (6.2b) and (6.2c) ensure that the scavenge variable ends at the start of the dispense variable, and the dispense variables ends at the start of the second temperature change variable, respectively. Constraints (6.2d) ensure that the values of the intervals of the gantry state function must always be equal to the value of the location of the corresponding project of the dispense intervals, and that these intervals are both left and right aligned. Likewise, constraints (6.2e) ensure that the values of the hazardous state function must always be equal to the medium value pressure during the scavenge intervals, and these intervals are both left and right aligned. Constraints (6.2f) ensure the dispense variables that require bulk chemicals connected to the gantry are considered by the gantry sequence variable, whereas constraints (6.2g) ensure the dispense variables requiring bulk chemicals attached to the bank robots are considered by the appropriate robot sequence variable. In both cases the type associated with the dispense variables indicates the location of the corresponding project. Constraints (6.2h) ensure the scavenge variables are considered by the hazardous sequence variable and all types indicate the medium vacuum pressure. Constraints (6.2i) and (6.2j) ensure that the first and second temperature change variable, respectively, increase the value of the power cumulative function by a unit for their durations. Finally, constraints (6.2k)-(6.2p) define the different interval variables considered.

All interval variables in the Bulk and None clusters can be expressed with respect to a single datum, $D_1^{Bulk/None}$.

Dispense Cluster

A Dispense Cluster, $(i, p, c) \in V^{Dispense}$, models a step in a staining or mixing protocol where a primary or mixed chemical is to be transferred to the either a slide or mixing vial. The Dispense Clusters are by far the most complicated of the interval clusters presented in this chapter. They combine aspects of the Bulk Clusters, and the model from the previous chapter (Section 5.5.2). Each Dispense Cluster consists of eight different types of interval variables set across four datums. Equivalent to the Bulk Cluster, the Dispense Cluster considers two temperature change intervals, $\alpha_{i,p,q}^{temp1}$ and $\alpha_{i,p,q}^{temp2}$, a scavenge interval, α^{scav}, and a dispense interval, α^{disp}. The lengths are determined as in the Bulk Cluster, with the exception that the length of the dispense variable is now d^{disp}. An aspirate interval variable, $\alpha_{i,p,q}^{aspirate}$ represents the aspirating of the appropriate chemical required by the cluster with fixed length $d^{aspirate}$. A wash interval variable, $\alpha_{i,p,q}^{wash}$, represents the washing of the pipette after the dispense of chemical. If $c \in C^{haz}$, then the hazardous vacuum is used for the wash and the interval has duration d^{wash2}, otherwise $c \in C^{safe}$ and the standard vacuum is used with duration d^{wash1}. A carry interval variable, $\alpha_{i,p,q}^{carry}$, is defined to represent the interval of time that the chemical that is dispensed during this cluster is being carried by the pipette. Similarly, a contaminate interval variable, $\alpha_{i,p,q}^{contam}$, is defined to represent the interval of time that the pipette is contaminated by the chemical that is dispensed during this cluster. The carry and contaminate intervals do not have a fixed length. The cluster level description of the Dispense Cluster can now be modelled as follows.

$$\texttt{endAtStart}(\alpha_{i,p,q}^{temp1}, \alpha_{i,p,q}^{disp}, d^{scav} \cdot \beta_{i,q}^{scav}), \qquad \forall (i,p,q) \in V^{Dispense} : \Delta_{i,q}^{temp1} > 0 \tag{6.3a}$$

$$\texttt{endAtStart}(\alpha_{i,p,q}^{scav}, \alpha_{i,p,q}^{disp}), \qquad \forall (i,p,q) \in V^{Dispense} : \beta_{i,q}^{scav} \tag{6.3b}$$

$$\texttt{endAtStart}(\alpha_{i,p,q}^{disp}, \alpha_{i,p,q}^{temp2}), \qquad \forall (i,p,q) \in V^{Dispense} : \Delta_{i,q}^{temp2} > 0 \tag{6.3c}$$

$$\texttt{startBeforeStart}(\alpha_{i,p,q}^{contam}, \alpha_{i,p,q}^{carry}), \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3d}$$

$$\texttt{startAtStart}(\alpha_{i,p,q}^{carry}, \alpha_{i,p,q}^{aspirate}), \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3e}$$

$$\texttt{endBeforeStart}(\alpha_{i,p,q}^{aspirate}, \alpha_{i,p,q}^{disp}, d_{i,p,q}^{A}), \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3f}$$

$$\texttt{endAtEnd}(\alpha_{i,p,q}^{disp}, \alpha_{i,p,q}^{carry}, d_{i,p,q}^{B}), \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3g}$$

$$\texttt{endBeforeStart}(\alpha_{i,p,q}^{disp}, \alpha_{i,p,q}^{wash}, d_{i,q}^{C}), \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3h}$$

$$\texttt{endAtEnd}(\alpha_{i,p,q}^{wash}, \alpha_{i,p,q}^{contam}), \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3i}$$

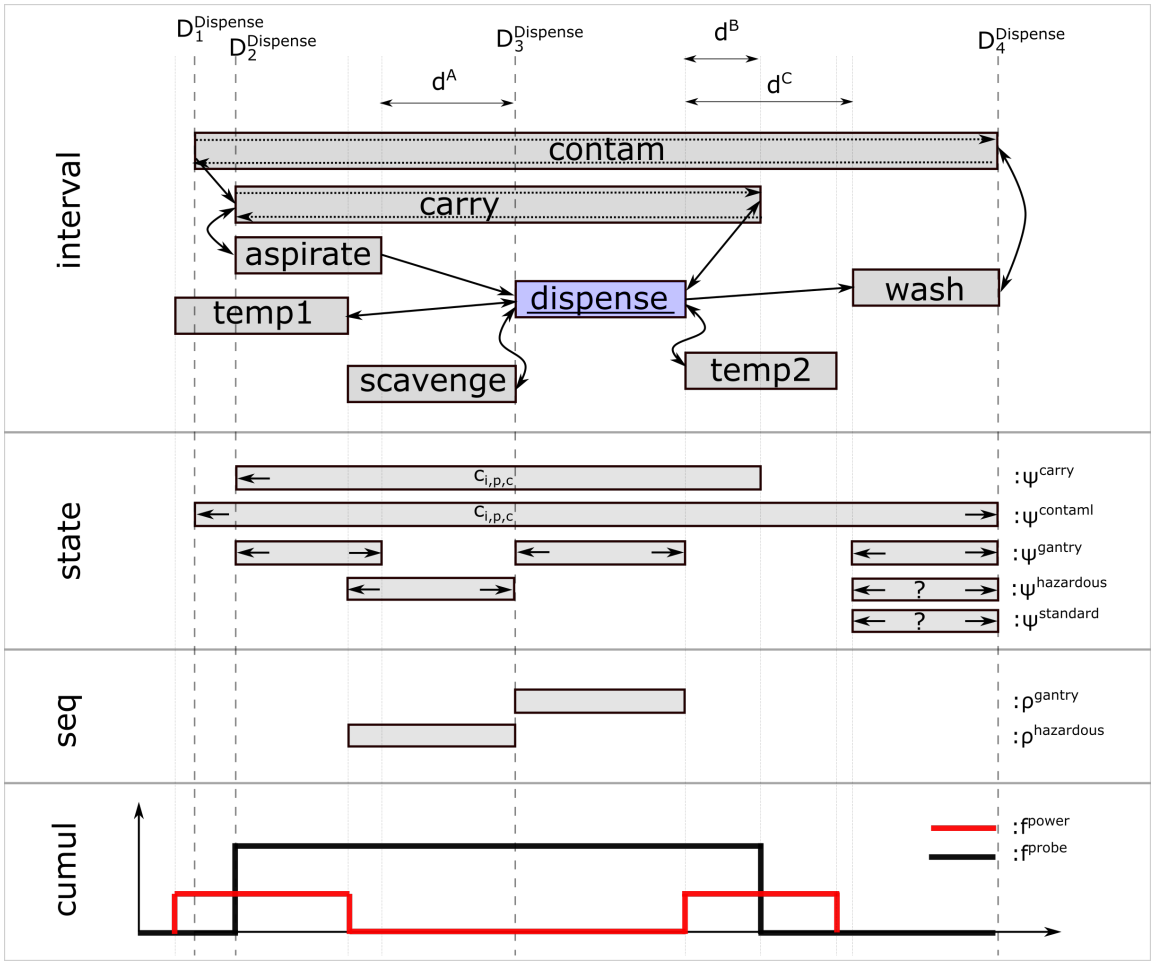


Figure 6.8: A visualisation of the HV Dispense cluster of interval variables and their interaction with the relevant sequence variables, state variables and cumul functions. Here the dispense variable is the driving variable. Note that the vacuum required during the wash interval depends on the chemical being dispensed, which is indicated by the "?" symbols.

$$\texttt{alwaysEqual}(\psi^{gantry}, \alpha^{aspirate}_{i,p,q}, L^{mix}, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Dispense} : c_{i,q} \in C^{mix} \tag{6.3j}$$

$$\texttt{alwaysEqual}(\psi^{gantry}, \alpha^{aspirate}_{i,p,q}, L^{vials}, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Dispense} : c_{i,q} \in C^{primary} \tag{6.3k}$$

$$\texttt{alwaysEqual}(\psi^{gantry}, \alpha^{disp}_{i,p,q}, l(p,q), \leftrightarrow) \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3l}$$

$$\texttt{alwaysEqual}(\psi^{gantry}, \alpha^{wash}_{i,p,q}, L^{wash}, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3m}$$

$$\texttt{alwaysEqual}(\psi^{haz}, \alpha^{scav}_{i,p,q}, \omega^{med}, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Dispense} : \beta^{scav}_{i,q} \tag{6.3n}$$

$$\texttt{alwaysEqual}(\psi^{haz}, \alpha^{wash}_{i,p,q}, \omega^{med}, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Dispense} : c_{i,q} \in C^{haz} \tag{6.3o}$$

$$\texttt{alwaysEqual}(\psi^{standard}, \alpha^{wash}_{i,p,q}, \omega^{med}, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Dispense} : c_{i,q} \in C^{standard} \tag{6.3p}$$

$$\texttt{alwaysEqual}(\psi^{carry}, \alpha^{carry}_{i,p,q}, c_{i,q}, \leftarrow) \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3q}$$

$$\texttt{alwaysEqual}(\psi^{contam}, \alpha^{contam}_{i,p,q}, c_{i,q}, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3r}$$

$$\texttt{inSequence}(\rho^{gantry}, \alpha^{disp}_{i,p,q}, L^{mix}) \qquad \forall (i,p,q) \in V^{Dispense} : c_{i,q} \in C^{mix} \tag{6.3s}$$

$$\texttt{inSequence}(\rho^{gantry}, \alpha^{disp}_{i,p,q}, l(p,q)) \qquad \forall (i,p,q) \in V^{Dispense} : c_{i,q} \in C^{primary} \tag{6.3t}$$

$$\texttt{inSequence}(\rho^{haz}, \alpha^{scav}_{i,p,q}, \omega^{med}) \qquad \forall (i,p,q) \in V^{Dispense} : \beta^{scav}_{i,q} \tag{6.3u}$$

$$f^{probe} \mathrel{+}= \texttt{pulse}(\alpha^{carry}_{i,p,q}, q(c_{i,q})) \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3v}$$

$$f^{power} \mathrel{+}= \texttt{pulse}(\alpha^{temp1}_{i,p,q}, 1) \qquad \forall (i,p,q) \in V^{Dispense} : \Delta^{temp1}_{i,q} > 0 \tag{6.3w}$$

$$f^{power} \mathrel{+}= \texttt{pulse}(\alpha^{temp2}_{i,p,q}, 1) \qquad \forall (i,p,q) \in V^{Dispense} : \Delta^{temp2}_{i,q} > 0 \tag{6.3x}$$

$$\texttt{presenceOf}(\alpha^{disp}_{i,p,q}) = \texttt{presenceOf}(\alpha^{var}_{i,p,q}) \qquad \forall (i,p,q) \in V^{Dispense}; var \in X^{disp} \tag{6.3y}$$

$$\alpha^{driving}_{i,p,q} = \alpha^{disp}_{i,p,q} \in \texttt{interval}(q \in Q^{stain}, p^{\downarrow}) \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3z}$$

$$\alpha^{aspirate}_{i,p,q} \in \texttt{interval}(q \in Q^{stain}, p^{wash}) \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3aa}$$

$$\alpha^{wash}_{i,p,q} \in \texttt{interval}(q \in Q^{stain}, d^{wash1}) \qquad \forall (i,p,q) \in V^{Dispense} : c_{i,q} \in C^{safe} \tag{6.3ab}$$

$$\alpha^{wash}_{i,p,q} \in \texttt{interval}(q \in Q^{stain}, d^{wash2}) \qquad \forall (i,p,q) \in V^{Dispense} : c_{i,q} \in C^{haz} \tag{6.3ac}$$

$$\alpha^{temp1}_{i,p,q} \in \texttt{interval}(q \in Q^{stain}, d^{temp} \cdot \Delta^{1}_{i,q}) \qquad \forall (i,p,q) \in V^{Dispense} : \Delta^{1}_{i,q} > 0 \tag{6.3ad}$$

$$\alpha^{temp2}_{i,p,q} \in \texttt{interval}(q \in Q^{stain}, d^{temp} \cdot \Delta^{2}_{i,q}) \qquad \forall (i,p,q) \in V^{Dispense} : \Delta^{2}_{i,q} > 0 \tag{6.3ae}$$

$$\alpha^{scav}_{i,p,q} \in \texttt{interval}(q \in Q^{stain}, d^{scav}) \qquad \forall (i,p,q) \in V^{Dispense} : \beta^{scav}_{i,q} \tag{6.3af}$$

$$\alpha^{carry}_{i,p,q} \in \texttt{interval}(q \in Q^{stain}, (d^{carry}_{i,p,q}, \infty)) \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3ag}$$

$$\alpha^{contam}_{i,p,q} \in \texttt{interval}(q \in Q^{stain}, (d^{contam}_{i,p,q}, \infty)) \qquad \forall (i,p,q) \in V^{Dispense} \tag{6.3ah}$$

Constraints (6.3a)-(6.3c) define the same end-at-start precedence constraints between the scavenge, dispense, and both temperature intervals as in the Bulk Dispense cluster. Constraints (6.3d)-(6.3i) are based on the precedence constraints (5.20d)-(5.20i) from the model presented in the previous chapter (Section 5.5.2). Constraints (6.3d) ensure the contaminate variables start before the carry variables start. Constraints (6.3e) ensure the aspirate and carry variables start together. Constraints (6.3f) ensure the ends of the aspirate variables leave sufficient time for the gantry to move to the correct location for the corresponding dispense variable, i.e., $d^{A}_{i,p,q} = d^{gantry}_{L^{aspirate},l(p,q)}$. Constraints (6.3g) ensure the ends of the dispense variables occur a specific amount of time before the end of the carry variables such that the gantry can move to the aspirate location, i.e., $d^{B}_{i,p,q} = d^{gantry}_{l(p,q),L^{aspirate}}$. Constraints (6.3h) ensure there is sufficient time after the end of the dispense variables before the start of the wash variables, i.e., $d^{C}_{i,p,q} = d^{gantry}_{l(p,q),L^{wash}}$. Constraints (6.3i) ensure the end of the wash variables and contaminate variables occur together.

Constraints (6.3j)-(6.3r) relate interval variables with the state functions. Constraints (6.3j)-(6.3m) enforce the correct travel times of the gantry are respected by ensuring the value of intervals of the gantry state function are always equal to the correct location for the aspirate variables for mixing projects, aspirate variables for

staining projects, dispense variables, and wash variables, respectively, and that the interval variables and the appropriate intervals from the gantry state function are both left and right aligned. Similarly, constraints (6.3n)- (6.3p) enforce the correct behaviour of the vacuums by ensuring the values of the intervals of the hazardous and standard state functions are always equal to medium pressure value for the scavenge and wash intervals. Constraints (6.3q) keep track of which chemical is currently in the pipette by ensuring the values of the intervals of the pipette state function are always equal to the correct chemical value during the carry intervals. Finally, constriants (6.3r) keep track of which chemical is contaminating the pipette by ensuring the values of the intervals of the chemical state function are always equal to the correct chemical value during the contaminate intervals.

Constraints (6.3s) and (6.3t) state that the dispense variables are considered by the gantry sequence variable and thus they are not allowed to overlap. Likewise, constraints (6.3u) state that the scavenge variables are considered by the hazardous sequence variable and thus are also not allowed to overlap. Constraints (6.3v) state that the probe cumulative function increases by a certain amount during the carry variables depending on the chemical required. Constraints (6.3w) and (6.3x) state that the power cumulative expression increases by a single unit during both the temperature variables. Constraints (6.3y) ensure that the presence status of the drive variables indicates the presence status of the other interval variables in the cluster, where $X^{disp} = (aspirate, wash, temp1, temp2, carry, contaminate, scavenge)$. Finally, constraints (6.3z)-(6.3ah) define the various interval variables.

The interval variables in a Dispense cluster can be partitioned according to the following datums:

- $D_1^{Dispense} = s(\alpha^{contam})$;
- $D_2^{Dispense} := \alpha^{aspirate}$ and $s(\alpha^{carry})$;
- $D_3^{Dispense} := \alpha^{disp}, \alpha^{temp1}, \alpha^{scav}, \alpha^{temp2}$, and $e(\alpha_{i,j}^{carry})$; and
- $D_4^{Dispense} := \alpha^{wash}$ and $e(\alpha^{contam})$.

Datum D_3^{disp} contains the driving variable, datums $D_1^{Dispense}$ and $D_2^{Dispense}$ are considered in $\mathcal{U}$, and datums $D_4^{Dispense}$ is considered in $\mathcal{L}$.

Load and Unload Clusters

A Load Cluster, $(i, p, q) \in V^{Load}$, represents the loading of a slide from the input drawer to the relevant unit and an Unload Cluster, $(i, p, q) \in V^{Unload}$, represents the unloading of a slide from the relevant unit to the output drawer. Firstly, both

clusters consider an up interval variable, $\alpha^{up}_{i,p,q}$, which represents the gantry picking up the slide from the correct location. Secondly, both clusters define a down interval variable, $\alpha^{down}_{i,p,q}$, which represents the gantry placing the slide at the correct location. Thirdly, both clusters consider a carry interval variable, $\alpha^{carry}_{i,p,q}$, which represents the interval of time the gantry is carrying the slide. Finally, the Unload Cluster considers a scavenge interval, $\alpha^{i,p,q}$, which represents the removal of the last remaining chemical on the slide before unloading.

The Load and Unload Clusters can now be modelled as follows.

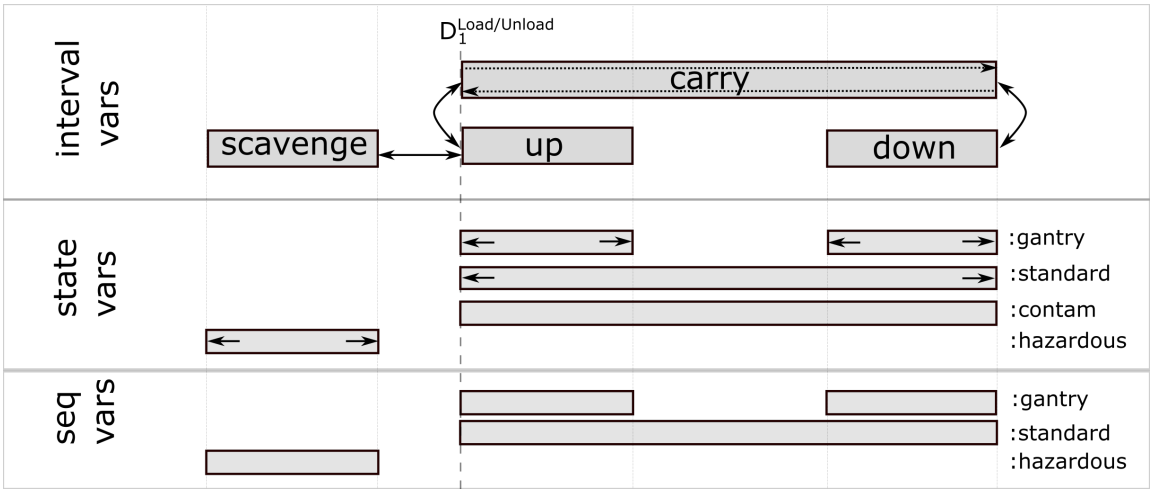


Figure 6.9: Visualisation of the Load and Unload Clusters. Scavenging is only considered when unloading

$$\texttt{startAtStart}(\alpha^{carry}_{i,p,q}, \alpha^{up}_{i,p,q}), \qquad \forall (i,p,q) \in V^{Load} \cup V^{Unload} \tag{6.4a}$$

$$\texttt{endAtEnd}(\alpha^{carry}_{i,p,q}, \alpha^{down}_{i,p,q}), \qquad \forall (i,p,q) \in V^{Load} \cup V^{Unload} \tag{6.4b}$$

$$\texttt{endAtStart}(\alpha^{scav}_{i,p,q}, \alpha^{up}_{i,p,q}) \qquad \forall (i,p,q) \in V^{Unload} \tag{6.4c}$$

$$\texttt{alwaysEqual}(\psi^{gantry}, \alpha^{up}_{i,p,q}, L^{input}, \leftrightarrow), \qquad \forall (i,p,q) \in V^{Load} \tag{6.4d}$$

$$\texttt{alwaysEqual}(\psi^{gantry}, \alpha^{up}_{i,p,q}, l(p,q), \leftrightarrow), \qquad \forall (i,p,q) \in V^{Unload} \tag{6.4e}$$

$$\texttt{alwaysEqual}(\psi^{gantry}, \alpha^{down}_{i,p,q}, l(p,q), \leftrightarrow), \qquad \forall (i,p,q) \in V^{Load} \tag{6.4f}$$

$$\texttt{alwaysEqual}(\psi^{gantry}, \alpha^{down}_{i,p,q}, L^{output}, \leftrightarrow), \qquad \forall (i,p,q) \in V^{Unload} \tag{6.4g}$$

$$\texttt{alwaysEqual}(\psi^{haz}, \alpha^{scav}_{i,p,q}, \omega^{med}, \leftrightarrow), \qquad \forall (i,p,q) \in V^{Unload} \tag{6.4h}$$

$$\texttt{alwaysEqual}(\psi^{standard}, \alpha^{carry}_{i,p,q}, \omega^{high}, \leftrightarrow), \qquad \forall (i,p,q) \in V^{Load} \cup V^{Unload} \tag{6.4i}$$

$$\texttt{alwaysNoState}(\psi^{contam}, \alpha^{carry}_{i,p,q}), \qquad \forall (i,p,q) \in V^{Load} \cup V^{Unload} \tag{6.4j}$$

$$\texttt{inSequence}(\rho^{gantry}, \alpha^{up}_{i,p,q}, L^{input}), \qquad \forall (i,p,q) \in V^{Load} \tag{6.4k}$$

$$\texttt{inSequence}(\rho^{gantry}, \alpha^{up}_{i,p,q}, l(p,q)), \qquad \forall (i,p,q) \in V^{Unload} \tag{6.4l}$$

$$\texttt{inSequence}(\rho^{gantry}, \alpha^{down}_{i,p,q}, l(p,q)), \qquad \forall (i,p,q) \in V^{Load} \tag{6.4m}$$

$$\texttt{inSequence}(\rho^{gantry}, \alpha^{down}_{i,p,q}, L^{input}), \qquad \forall (i,p,q) \in V^{Unload} \tag{6.4n}$$

$$\texttt{inSequence}(\rho^{haz}, \alpha^{scav}_{i,p,q}, \omega^{med}), \qquad \forall (i,p,q) \in V^{Unload} \tag{6.4o}$$

$$\texttt{inSequence}(\rho^{standard}, \alpha^{carry}_{i,p,q}, \omega^{high}), \qquad \forall (i,p,q) \in V^{Load} \cup V^{Unload} \tag{6.4p}$$

$$\alpha^{driving}_{i,p,q} = \alpha^{up}_{i,p,q} \in \texttt{interval}(comp, d^{pickup}), \qquad \forall (i,p,q) \in V^{Load} \tag{6.4q}$$

$$\alpha^{driving}_{i,p,q} = \alpha^{up}_{i,p,q} \in \texttt{interval}(comp, d^{unload}) \qquad \forall (i,p,q) \in V^{Unload} \tag{6.4r}$$

$$\alpha^{down}_{i,p,q} \in \texttt{interval}(comp, d^{load}), \qquad \forall (i,p,q) \in V^{Load} \tag{6.4s}$$

$$\alpha^{down}_{i,p,q} \in \texttt{interval}(comp, d^{place}) \qquad \forall (i,p,q) \in V^{Unload} \tag{6.4t}$$

$$\alpha^{carry}_{i,p,q} \in \texttt{interval}(comp, d^{carry}_{p,q}), \qquad \forall (i,p,q) \in V^{Load} \cup V^{Unload} \tag{6.4u}$$

$$\alpha^{scav}_{i,p,q} \in \texttt{interval}(comp, d^{scav}) \qquad \forall (i,p,q) \in V^{Unload} \tag{6.4v}$$

Constraints (6.4a) and(6.4b) ensure the carry intervals start at the start of the corresponding up interval and end at the end of the corresponding down interval, respectively. Constraints (6.4c) ensure the scavenge intervals end at the start of the corresponding up interval. Constraints (6.4d)-(6.4g) enforce the correct locations of the gantry by ensuring the values of the intervals of the gantry state function are always equal to the correct locations for the up and down intervals. Constraints (6.4k)-(6.4n) pass the same information to the gantry sequence variable. Likewise, constraints (6.4i) ensure the values of the intervals of the standard vacuum state function are always equal to the high pressure value during the carry intervals, and constraints (6.4p) pass this information to the standard vacuum sequence variable. Constraints (6.4h) enforce the hazardous vacuum has the correct pressure by ensuring the values of the intervals from the hazardous state function are always equal to the medium pressure value during the scavenge intervals, and constraints (6.4o) pass this same information to the hazardous sequence variable. Constraint (6.4j) enforces that the pipette of the gantry is clean by ensuring the chemical state function has no state during the carry intervals. Finally, constraints (6.4q)-(6.4v) define the various interval variables of the cluster.

All interval variables in the Load and Unload clusters can be expressed with respect to a single datum, $D_1^{Load/Unload}$.

Mixing and Cleaning Clusters

A Mixing Cluster, $(i,p,q) \in V^{mix}$, represents the procedure of encouraging chemicals in a mixing vial to react by having the pipette of the gantry blow bubbles. A

Cleaning Cluster, $(i, p, q) \in V^{Clean}$, represents the cleaning of the mixing vial by the pipette of the gantry.[4] Both clusters consider three types of interval variables. Firstly, a vial interval variable, $\alpha^{vial}_{i,p,q}$, is defined to represent either the mixing or the cleaning at the vial by the pipette of the gantry. For the Clean cluster the duration of the vial interval is d^{clean}, whereas for the Mixing cluster the duration is d^{mix}. Secondly, a wash interval variable is defined to represent the cleaning of the pipette at the wash station. Similar to other clusters, if a hazardous chemical is used the interval has d^{haz}, whereas if a safe chemical is used the interval has d^{safe}. Finally, a contaminate interval variable, $\alpha^{contam}_{i,p,q}$, is defined to represent the time that the specific mix of clean contaminates the pipette. The Mixing and Cleaning Clusters can now be modelled as follows.

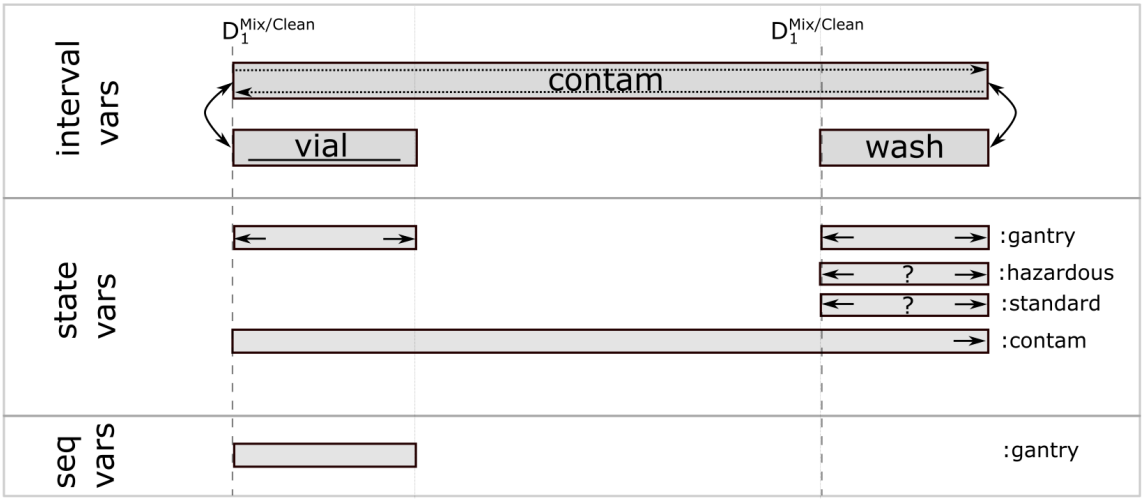


Figure 6.10: A visualisation of the Mix / Clean Cluster of interval variables

$$\texttt{startAtStart}(\alpha^{contam}_{i,p,q}, \alpha^{vial}_{i,p,q}) \qquad \forall (i,p,q) \in V^{Mix} \cup V^{Clean} \tag{6.5a}$$

$$\texttt{endAtEnd}(\alpha^{contam}_{i,p,q}, \alpha^{wash}_{i,p,q}) \qquad \forall (i,p,q) \in V^{Mix} \cup V^{Clean} \tag{6.5b}$$

$$\texttt{alwaysEqual}(\psi^{gantry}, \alpha^{vial}_{i,p,q}, L^{mix}, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Mix} \cup V^{Clean} \tag{6.5c}$$

$$\texttt{alwaysEqual}(\psi^{gantry}, \alpha^{wash}_{i,p,q}, L^{wash}, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Mix} \cup V^{Clean} \tag{6.5d}$$

$$\texttt{alwaysEqual}(\psi^{haz}, \alpha^{wash}_{i,p,q}, \omega^{med}, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Mix} \cup V^{Clean} : c_{i,q} \in C^{haz} \tag{6.5e}$$

$$\texttt{alwaysEqual}(\psi^{standard}, \alpha^{wash}_{i,p,q}, \omega^{med}, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Mix} \cup V^{Clean} : c_{i,q} \in C^{safe} \tag{6.5f}$$

$$\texttt{alwaysEqual}(\psi^{contam}, \alpha^{contam}_{i,p,q}, c_{i,q}, \rightarrow) \qquad \forall (i,p,q) \in V^{Mix} \cup V^{Clean} : c_{i,q} \in C^{safe} \tag{6.5g}$$

$$\texttt{inSequence}(\rho^{gantry}, \alpha^{vial}_{i,p,q}, L^{mix}) \qquad \forall (i,p,q) \in V^{Mix} \cup V^{Clean} \tag{6.5h}$$

$$\texttt{presenceOf}(\alpha^{vial}_{i,p,q}) = \texttt{presenceOf}(\alpha^{var}_{i,p,q}) \qquad \forall (i,p,q) \in V^{Mix} \cup V^{Clean}; var \in X^{mix} \tag{6.5i}$$

[4] In reality cleaning might require multiple protocol steps but for simplicity we are just assuming it takes a single interval cluster.

$$\alpha^{driving}_{i,p,q} = \alpha^{vial}_{i,p,q} \in \texttt{interval}(opt, d^{mix}) \qquad \forall (i,p,q) \in V^{Mix} \tag{6.5j}$$

$$\alpha^{driving}_{i,p,q} = \alpha^{vial}_{i,p,q} \in \texttt{interval}(opt, d^{clean}) \qquad \forall (i,p,q) \in V^{Clean} \tag{6.5k}$$

$$\alpha^{wash}_{i,p,q} \in \texttt{interval}(opt, d^{wash1}) \qquad \forall (i,p,q) \in V^{Mix} \cup V^{Clean} : c_{i,q} \in C^{safe} \tag{6.5l}$$

$$\alpha^{wash}_{i,p,q} \in \texttt{interval}(opt, d^{wash2}) \qquad \forall (i,p,q) \in V^{Mix} \cup V^{Clean} : c_{i,q} \in C^{haz} \tag{6.5m}$$

$$\alpha^{contam}_{i,p,q} \in \texttt{interval}(opt, (d^{contam}_{i,p,q}, \infty)) \qquad \forall (i,p,q) \in V^{Mix} \cup V^{Clean} \tag{6.5n}$$

Constraints (6.5a) and (6.5b) ensure the contaminate variables start at the start of the corresponding mix variable and end at the end of the corresponding wash variable, respectively. Constraints (6.5c) and (6.5d) account for the travel times between locations by ensuring the values of the intervals of the gantry state functions are always equal to the location of the mixing vials during the mix intervals, and the location of the washing station during the wash intervals, respectively. Constraints (6.5e) ensure the values of the intervals of the hazardous state function are always equal to the medium pressure level during the wash intervals for activities requiring hazardous chemicals, whereas constraints (6.5f) ensure the same thing for the standard state function for safe chemicals. Constraints (6.5g) represents the contamination of the pipette being mixed by ensuring the values of the intervals of the chemical state function are always equal to the index of the chemical being mixed for the duration of the contaminate interval. Constraints (6.5h) state that the mix intervals are considered by the gantry sequence variable and the types associated with the mix intervals correspond to the location of the mixing vials. Constraints (6.5i) ensure that the presence status of the drive variables indicates the presence statuses of the other interval variables in the cluster, where $X^{mix} = (contaminate, wash)$. Finally, constraints (6.5j)-(6.5n) define the interval variables of the cluster. Here the minimum contamination time, $d^{contam}_{i,p,q}$, is equal to appropriate the d^{mix} or d^{clean} plus the transit time $d^{gantry}_{L^{mixing},L^{wash}}$ plus the appropriate washing time d^{wash1} or d^{wash2}.

The interval variables in the Mix and Clean clusters can be partitioned according to the following datums:

- $D^{Mix/Clean}_1 = \alpha^{vial}$ and $s(\alpha^{contam})$; and
- $D^{Mix/Clean}_2 = \alpha^{wash}$ and $e(\alpha^{contam})$.

Datum $D^{Mix/Clean}$ contains the driving variable and datum $D^{Mix/Clean}_2$ is considered in $\mathcal{L}$.

Agitate Cluster

An Agitate Cluster, $(i, p, q) \in V^{Agitate}$, represents the processing of accelerating a chemical reaction by moving a chemical back and forth across the sample using the appropriate bank robot. The clusters consider two types of interval variables. Firstly, an agitate interval variable, $\alpha^{agitate}_{i,p,q}$, is defined with fixed duration $d^{agitate}$ representing the time the appropriate bank robot is at the slide agitating the reaction. Secondly, a wash interval, α^{wash}, is defined with fixed duration d^{wash3}, representing the bank robot washing its probe at the appropriate wash station. The Agitate Cluster can now be modelled as follows.

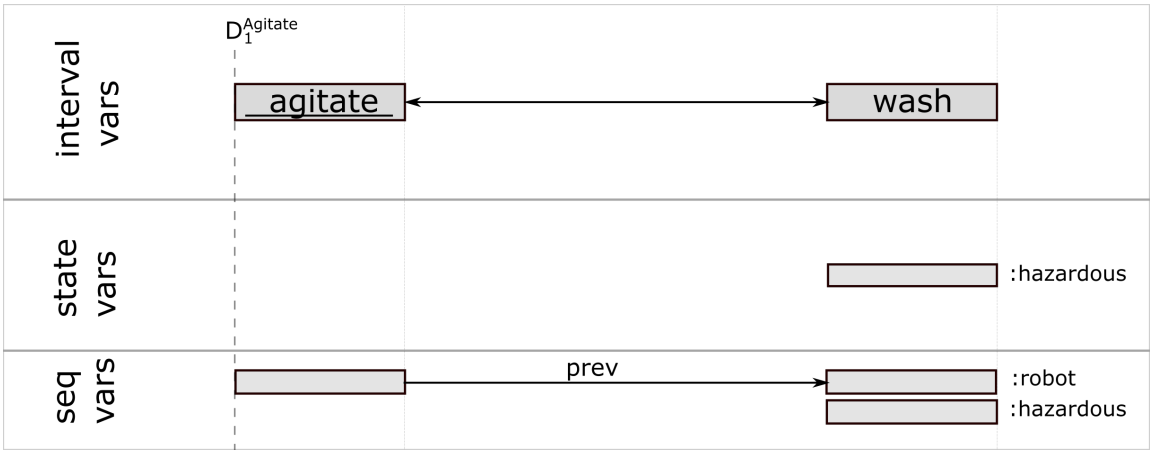


Figure 6.11: A visualisation of the Agitate cluster of interval variables and their interaction with the relevant sequence and state variables. Here the agitate variable is considered the driving variable.

$$\texttt{endAtStart}(\alpha^{agitate}_{i,p,q}, \alpha^{wash}_{i,p,q}, d_{p,q}) \qquad \forall (i,p,q) \in V^{Agitate} \tag{6.6a}$$

$$\texttt{alwaysEqual}(\psi^{haz}, \alpha^{wash}_{i,p,q}, \omega^{med}, -) \qquad \forall (i,p,q) \in V^{Agitate} \tag{6.6b}$$

$$\texttt{inSequence}(\rho^{haz}, \alpha^{wash}_{i,p,q}, \omega^{med}) \qquad \forall (i,p,q) \in V^{Agitate} \tag{6.6c}$$

$$\texttt{inSequence}(\rho^{robot}_{b(p,q)}, \alpha^{agitate}_{i,p,q}, L^{mix}) \qquad \forall (i,p,q) \in V^{Agitate} \tag{6.6d}$$

$$\texttt{inSequence}(\rho^{robot}_{b(p,q)}, \alpha^{wash}_{i,p,q}, L^{wash}_{b(p,q)}) \qquad \forall (i,p,q) \in V^{Agitate} \tag{6.6e}$$

$$\texttt{prev}(\rho^{robot}_{b(p,q)}, \alpha^{agitate}_{i,p,q}, \alpha^{wash}_{i,p,q},) \qquad \forall (i,p,q) \in V^{Agitate} \tag{6.6f}$$

$$\alpha^{driving}_{i,p,q} = \alpha^{agitate}_{i,p,q} \in \texttt{interval}(comp, d^{agitate}) \qquad \forall (i,p,q) \in V^{Agitate} \tag{6.6g}$$

$$\alpha^{wash}_{i,p,q} \in \texttt{interval}(comp, d^{wash3}) \qquad \forall (i,p,q) \in V^{Agitate} \tag{6.6h}$$

Constraints (6.6a) enforce that the end of the agitate variable occurs exactly the amount of time to travel from the location of the slide to the appropriate bank

robot wash station before the start of the wash station, i.e., $d_{p,q} = d^{robot}_{b(p,q),l(p,q),L^{wash}_{b(p,q)}}$. Constraints (6.6b) ensures the values of the intervals from the hazardous state function are always equal to the medium pressure value during the wash intervals. Similarly, constraints (6.6c) tell this information to the hazardous sequence variable. Constraints (6.6d) and (6.6e) state that the agitate and wash variables are included in the appropriate bank robot sequence variables, respectively, where the types associated with the variables indicate the location of the intervals. Constraints (6.6f) ensure the the agitate variables occur immediately before their corresponding wash variables in the appropriate bank robot sequence variables. Finally, constraints (6.6g) and (6.6h) define the agitate and wash variables respectively.

All interval variables in the Agitate clusters can be expressed with respect to a single datum.

Microscavenge Cluster

A Micro Cluster, $(i, p, q) \in V^{micro}$, represents the process, known as a microscavenge, of removing bubbles from an individual unit through the application of the hazardous vacuum. The cluster consists of a single interval variable, α^{micro}, the has a fixed duration of d^{micro}. The Micro Cluster is modelled as follows.

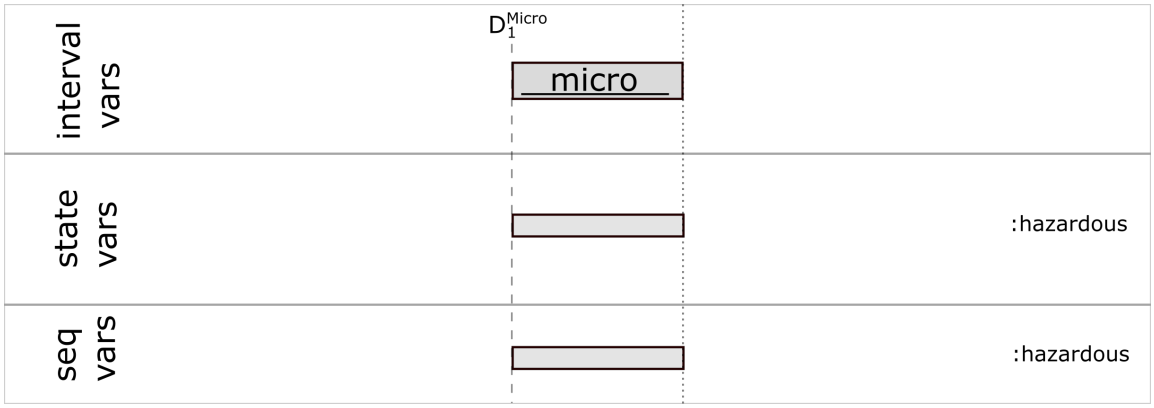


Figure 6.12: A visualisation of the Microscavenge cluster, which contains a single interval variable.

$$\texttt{alwaysEqual}(\psi^{haz}, \alpha^{micro}_{i,p,q}, \omega^{low}, -) \qquad \forall (i,p,q) \in V^{Micro} \tag{6.7a}$$

$$\texttt{inSequence}(\rho^{haz}, \alpha^{micro}_{i,p,q}, \omega^{low}) \qquad \forall (i,p,q) \in V^{Micro} \tag{6.7b}$$

$$\alpha^{driving}_{i,p,q} = \alpha^{micro}_{i,p,q} \in \texttt{interval}(comp, d^{micro}) \forall (i,p,q) \in V^{Micro} \tag{6.7c}$$

Constraints (6.7a) ensure the values of the intervals of the hazardous state function are always equal to the low vacuum pressure during the micro intervals. Similarly, constraints (6.7b) tells this information to the hazardous sequence variable. Finally, constraints (6.7c) simply define the micro interval variable. All intervals in the Micro Cluster can be expressed with respect to a single datum.

Dummy Cluster

A Dummy Cluster, $(i, p, q) \in V^{Dummy}$, represent possible dispenses of a specific mixed chemical from a specific mixing project. Dummy Clusters from the mixing projects are mapped to Dispense Clusters from the staining projects requiring that mixed chemical by `isomorphism` constraints. The Dummy Cluster consists of two interval variables. Firstly, like Dispense Clusters, a dispense interval variable, $\alpha^{disp}_{i,p,q}$. Secondly, a contaminate interval variable, $\alpha^{contam}_{i,p,q}$, which again represents the relevant interval of time where the pipette is contaminated. The Dummy Clusters are modelled as followed.

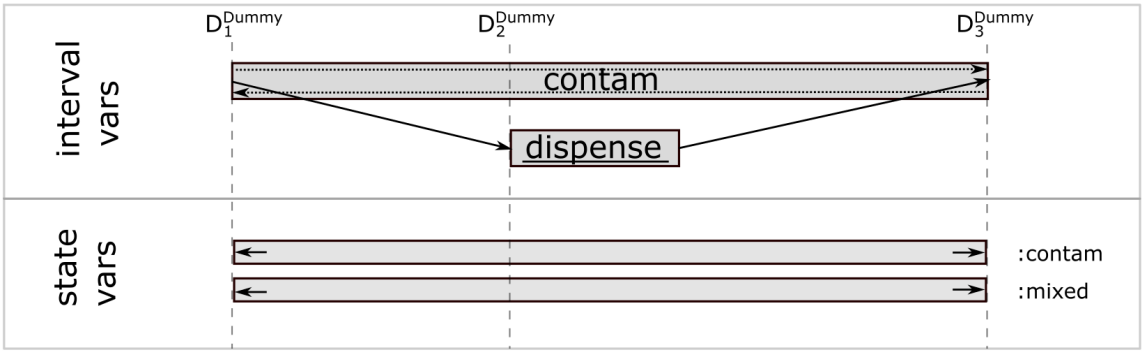


Figure 6.13: A visualisation of the Dummy Dispense cluster of interval variables and their interaction with the relevant state variables. Here the dispense variable is the driving variable.

$$\texttt{startBeforeStart}(\alpha^{contam}_{i,p,q}, \alpha^{disp}_{i,p,q}, d^{A}_{i,p,q}), \qquad \forall (i,p,q) \in V^{Dummy} \tag{6.8a}$$

$$\texttt{endBeforeEnd}(\alpha^{disp}_{i,p,q}, \alpha^{contam}_{i,p,q}, d^{C}_{i,p,q}), \qquad \forall (i,p,q) \in V^{Dummy} \tag{6.8b}$$

$$\texttt{alwaysEqual}(\psi^{contam}, \alpha^{contam}_{i,p,q}, c_{i,q}, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Dummy} \tag{6.8c}$$

$$\texttt{alwaysEqual}(\psi^{mixing}, \alpha^{contam}_{i,p,q}, p, \leftrightarrow) \qquad \forall (i,p,q) \in V^{Dummy} \tag{6.8d}$$

$$\texttt{presenceOf}(\alpha^{disp}_{i,p,q}) = \texttt{presenceOf}(\alpha^{contam}_{i,p,q}) \qquad \forall (i,p,q) \in V^{Dummy} \tag{6.8e}$$

$$\alpha^{driving}_{i,p,q} = \alpha^{disp}_{i,p,q} \in \texttt{interval}(opt, d^{disp}) \qquad \forall (i,p,q) \in V^{Dummy} \tag{6.8f}$$

$$\alpha^{contam}_{i,p,q} \in \texttt{interval}(opt, (d^{contam}_{i,p,q}, \infty)) \qquad \forall (i,p,q) \in V^{Dummy} \tag{6.8g}$$

Constraints (6.8a) and (6.8b) ensure the contaminate variables start before the start, and end after the end of the corresponding dispense variables. The distances used in these precedences constraints come from the minimum lengths of the contaminate variables from the dispense clusters requiring the mixed reagent, i.e., for operation $(i,p,q) \in V^{Dummy}$ the lengths are $d^A_{i,p,q} = p^{\downarrow} + \min_{(i',p',q') \in V^{Dispense}: c_{i,q} = c_{i',q'}} d^A_{i',p',q'}$ and $d^C_{i,p,q} = p^{wash} + \min_{(i',p',q') \in V^{Dispense}: c_{i,q} = c_{i',q'}} d^C_{i',p',q'}$. Constraints (6.8c) ensure the values of intervals of the chemical state function are always equal to the index of the mixed chemical that is being dispensed. Furthermore as the intervals from the chemical state function and the contaminate intervals are both left and right aligned, and due to the isomorphism constraints, if a dispense from a dummy cluster is mapped to a dispense from a dispense cluster, the corresponding contaminate variables will also become aligned. Constraints (6.8d) state that mixed chemical cannot be used from two different projects, i.e., mixed chemical cannot be aspirated from two different vials. This is completed by ensuring the values of intervals from the mixed state function are always equal to the index of the project from the mixing class for the duration of the contaminate variable. Constraints (6.8e) ensure the presence status of the dispense variable dictates the presence status of the contaminate variable. Finally, constraints (6.8f) and (6.8g) define the interval variables from the cluster and state that the dispense variable is the driving variable.

The interval variables in the Dummy clusters can be partitioned according to the following datums:

- $D_1^{Dummy} = s(\alpha^{contam})$;
- $D_2^{Dummy} = \alpha^{disp}$; and
- $D_3^{Dummy} = e(\alpha^{contam})$.

Datum D_2^{Dummy} contains the driving variable, datum D_1^{Dummy} is considered in $\mathcal{U}$, and D_3^{Dummy} is considered in $\mathcal{L}$.

6.5 Computational Study

In this section we provide a brief computational study to demonstrate the effectiveness of solving the model with CP Optimizer 12.8.

6.5.1 Data

To evaluate the model we consider twenty-four instances based on real-world data. Each of the instances consider the simple scenario where the instrument is

completely empty and then twenty-four slides are loaded into the input drawer and a schedule is required. Half of the instances consider the case where all of the staining protocols are the same, where the other half randomly selects a set of twenty four projects from the 12 different staining protocols.

Protocols

We consider the two main types of mixed chemicals referred to as *DAB* and *Red*. The mixing protocol for DAB creates up to three units that must be consumed within an hour. The mixing protocol for Red creates up to two units that must be consumed within five minutes. The mixing protocol for DAB consists of combining two component chemicals, whereas for Red it consists of four. For these reasons, DAB can be thought of as the easier of the two mixed chemicals.

The instances created consider up to twelve different staining protocols. The staining protocols are classified based on the types of mixed chemical they require (Red, DAB, Double), the scientific process used to expose target cells (EIER, HIER), and whether agitation and microscavenging is required, indicated with a plus sign (+). For example, the protocol denoted *DAB-EIER+* requires just DAB, uses EIER to expose target sites and does consider agitation and microscavenging. Each staining protocol consists of approximately 50 steps with between 5-10 of those steps requiring a primary chemical, 1-2 requiring a mixed chemical, and the rest requiring bulk chemicals. The minimum waiting time of each step is often zero after dispensing a bulk chemical, and range from 3 minutes to 30 minutes after HV chemicals. The maximum waiting time of each step is typically 30 seconds to 1-2 minutes more than the minimum time, but in a few of the steps can be up to half an hour. Typically the maximum times are stricter for the HV chemicals. The science time of staining protocols typically range from one to two hours.

Instrument

The instrument consists of twenty-four units divided across two banks. The various durations for tasks such as aspirating, dispensing and mixing, range from two to eight seconds. The time the gantry and robots require to move between locations can take up to three seconds. The number of mixing vials exceeds what is required in the experiments so this does not need to be considered. The capacity of the pipette can take up to six units of certain primary and mixed chemicals at a time but more commonly can only take two to three units. In order to ensure the power budget is not exceeded, only six different units can be heated or cooled at the same

time.

6.5.2 Model Improvements

In order to strengthen the model, it is possible to provide additional information to the solver. This is typically done by providing as much information to the temporal network as possible. For example, a protocol might specify that no waiting time is required between two steps, yet due to the physical set up of the system it might not be possible to start the next step immediately. This information can be passed to the model in two ways. Firstly, by strengthening the minimum time lags between driving variables of clusters. Secondly, by considering additional precedences between auxiliary variables. Strengthening methods like this have been considered in the previous two chapters of the thesis. In implementing the model, we considered each pair of cluster types for which one could potentially precede the other explicitly. For each pair we added as much additional information to the solver as possible. This resulted in many different types of constraints that we will omit from the thesis for brevity.

6.5.3 Search Phases

Driving Variables and LU Consistency

The benefit of modelling the problem in terms of interval clusters is that we consider driving and auxiliary variables while modelling. Again we pass this information to the solver through search phases. The first search phase only consider the driving variables of the problem. The second search phase considers a single variable associated with each datum assigned to $\mathcal{U}$, i.e., $\{D_1^{Dummy}, D_1^{Dispense}, D_2^{Dispense}\}$, randomly selects a variable and assigns it to its current upper bound. The third search phase considers a single variable associated with each datum assigned to $\mathcal{L}$, i.e., $\{D_3^{Dummy}, D_4^{Dispense}, D_2^{Mix/Clean}\}$, randomly selects a variable and assigns it to its current lower bound. As CP Optimizer does not allow for interval variables to be set to their latest start time, additional integer variables are created and used for the second search phase as was required in the previous two chapters.

List Heuristic

Even when solving the model with search phases to implement the driving and auxiliary variables correctly, for real-world sized instances the default search of CP Optimizer was not able to find an initial solution within a reasonable amount

of time. To overcome this we considered an additional set of search phases and constraints to effectively force CP Optimizer to implement a list heuristic.

Firstly additional constraints are added to the model to essentially fix the isomorphism constraint (6.1f). Each dispense cluster from a staining class that required a mixed chemical, is mapped to the first dummy dispense from a unique project from the mixing class that produced the required mixed chemical. Let $map(i,p,q)$ be the dummy cluster of the project from the mixing class that is mapped from activity (i,p,q) of the staining project from the staining class. The following constraints are then considered,

$$\texttt{presenceOf}(\alpha^{disp}_{map(i,p,q)}) == 1 \qquad \forall (i,p,q) \in V : c_{i,q} \in C^{mix} \wedge q \in Q^{staining} \tag{6.9a}$$

$$\texttt{startAtStart}(\alpha^{disp}_{i,p,q}, \alpha^{disp}_{map(i,p,q)}) \qquad \forall (i,p,q) \in V : c_{i,q} \in C^{mix} \wedge q \in Q^{staining} \tag{6.9b}$$

Constraints (6.9a) ensure the first dummy dispense from each of the projects from the mixing classes is present. Through propagation of the presence network and with the isomorphism constriant, this forces all mixing projects to be present and the remaining dummy variables to be absent. Constraints (6.9b) then ensure the appropriate mapping between staining. Hence we simply assume that each time a mixed chemical is required it is produced by its own mixing project.

The list heuristic is based on the idea that staining projects can be inserted into the schedule one at a time. From a mathematical point of view, it is based on the contracted graphs introduced by Neumann and Zhan (1995) and that is discussed in Chapter 2, where arcs of strongly connected components are contracted and then the resulting nodes inserted into the schedule in the order they are considered in the contracted graph. The order of the list is determined by Algorithm 5.

The list heuristic is passed to CP Optimizer through a large number of search phases. A single search phase is added that contains each driving variable in the order they are found in the list. Once all of the driving variables are added again a search phase is added for the $\mathcal{U}$ variables and a search phase is added for the $\mathcal{L}$ variables.

6.5.4 Experimental Setup

The approach evaluated in the preliminary computational study presented in this chapter is based on combining the two different types of search phases. Firstly

Algorithm 5 The order of the list heuristic

1: **for** each staining class $q \in Q^{staining}$ **do**
2: **for** each project $p \in P_q$ **do**
3: **for** each cluster $i \in V_q$ **do**
4: **if** $(i, p, q) \in V^{Disp} \wedge c_{i,q} \in C^{mix}$ **then**
5: Consider mapped cluster, $(i', p', q') = map(i, p, q)$, and insert all of the driving variables from clusters (i'', p', q') where $i'' \leq i'$ into the list in increasing order of the cluster id.
6: The driving variable from cluster (i, p, q) is inserted in the list.
7: **for** each mixing class $q \in Q^{mixing}$ **do**
8: **for** each project $p \in P_q$ **do**
9: Insert the last cluster into the list

we aim to quickly find a reasonable first solution. To implement this, we consider the additional constraints, (6.9a) and (6.9b), and implement the list heuristic. CP Optimizer is given a single worker at this stage. We wish to restrict CP Optimizer to one worker so that we could potentially run the list heuristic in parallel with different list orderings in the future. Secondly, we aim to see whether the default search of CP Optimizer can improve this solution. Hence we provide the solution obtained by the list heuristic to CP Optimizer as a feasible solution. At this stage we also remove the additional constraints, (6.9a) and (6.9b), from the model. CP Optimizer then performs the default search with three workers but still considering the search phases to implement the driving variables correctly. This approach is evaluated on the described twenty-four real-world instances.

6.5.5 Computational Results

A summary of the computational results is given in Table 6.5. The columns of the table record the following results. Firstly the quality of the initial solution is evaluated based on how many times the solver had to back-track during search before finding a feasible solution (Fails), the CPU time required to find the feasible solution, and the objective function. Recall that the objective function is the makespan of the schedule, presented here in seconds. Once the initial solution is seeded into the complete model, we measure the quality of the objective function after 1 minute and 10 minutes. This is reported as a optimality gap taken with the best lower bound in the table in the appropriate (Gap) columns. Finally the best lower bound (LB) obtained by CP Optimizer for the complete model and the lower bound obtained from the initial propagation of the temporal network (LB-T) are

given. The lower bound obtained from the temporal network can be interpreted as the maximum amount of time required to schedule a single staining project optimally on its own.

Table 6.5: Experimental results

Inst.	Name	Initial Sol.			1 min	10 min	LB	LB-T
		Fails	Time (s)	Gap	Gap	Gap		
1	DAB-HIER	73	2.63	19.1	3.2	1.9	5659	4792
2	DAB-EIER	4	7.24	47.0	7.2	6.0	3734	3294
3	DAB-HIER+	80	3.16	22.3	5.7	2.1	5659	4792
4	DAB-EIER+	3	3.32	60.8	7.1	4.8	3734	3294
5	Red-HIER	45	3.3	14.7	12.0	5.1	7513	6646
6	Red-EIER	154	3.24	13.9	13.3	10.1	5588	5148
7	Red-HIER+	23	3.53	12.0	9.0	5.3	7513	6646
8	Red-EIER+	127	3.57	13.9	12.9	10.6	5588	5148
9	Double-HIER	63	6.47	12.4	11.1	10.7	7891	7024
10	Double-EIER	34	12.09	37.1	35.5	34.1	5966	5526
11	Double-HIER+	63	8.48	12.4	11.2	10.9	7891	7024
12	Double-EIER+	34	8.21	37.1	34.6	30.6	5966	5526
13	Combination-1	28	4.14	12.2	12.0	6.6	7046	7022
14	Combination-2	92	4.2	26.4	26.0	10.5	7058	7022
15	Combination-3	38	3.37	15.7	15.7	14.7	7036	7022
16	Combination-4	293	4.15	13.2	13.0	9.4	7172	7022
17	Combination-5	111	3.19	12.5	12.5	4.5	7069	7022
18	Combination-6	21	3.44	15.6	15.6	7.4	7058	7022
19	Combination-7	32	8.46	15.6	13.8	0.1	7058	7022
20	Combination-8	115	3.29	13.3	11.6	2.2	7172	7022
21	Combination-9	42	4.05	12.9	11.2	7.1	7069	7022
22	Combination-10	34	4.58	17.3	17.3	9.2	7069	7022
23	Combination-11	34	3.03	13.4	12.0	8.9	7058	7022
24	Combination-12	81	3.11	17.7	16.0	5.1	7022	7022

The study successfully demonstrates the validity of our approach. In all of the instances, our list heuristic is able to obtain reasonable feasible solutions very quickly, i.e., in approximately 5 seconds. Recall that in practice a schedule must be generated within a minute such that the machine can begin processing, which the list heuristic clearly satisfies. In fact, it would even be possible to run variants of this heuristic to obtain a pool of starting solutions. Furthermore, the majority of solutions can be improved to within 10% optimality gap, which we consider to very efficient. Noticeably larger gaps are reported for instances 10 and 12, both consisting of a single staining protocol consisting of both mixed chemicals and

using EIER for epitope retrieval. These instances have a higher demand on the resources and thus we suspect the lower bound might be improved by explicitly considering a resource-based lower bound similar to the previous chapter.

The speed with which initial solutions can be found is a immediate result of how few fails occur when performing the list heuristic. Many of the instances obtain a feasible solution with less than 100 fails and the worst occurrence is in instance 16 with 293 fails. When considering that the models have approximately 4,000 variables and 12,000 constraints, and many tight maximum time lags, we were impressed by how few fails occurred. We believe this result reflects that the constraints are propagating efficiently and as intended. Based on these results however the direct relationship between the number of fails and time taken to obtain a result is not immediately clear. We wish to investigate this further in future tests.

The lower bounds obtained by propagating the temporal network reveal the need for further testing. All twelve of the instances created from combinations have the same $LB - T$. This indicates that all of the instances contain at least one slide that requires a Double-HIER(+) staining protocol. Given that we are only considering the makespan of the problem, the protocols that require the most amount of time largely dictate the time required to complete a group of instances. In the future it could be beneficial to evaluate the instances on average project delay to get a more insightful understanding.

The full model improves the initial solutions with varying degrees of success. In some instances, e.g. instances 1, 2, and 4, it is possible to significantly improve the objective within 1 minute. For some instances, e.g. instances 5, 14, 18, and 19, there is very little improvement within 1 minute but considerable improvement after 10. For some instances, e.g., 9, 11, and 15, there is very little improvement at all. Regardless, the experimental results sufficiently demonstrate that it can beneficial to try to improve the initial solution and that this can be achieved in a relatively short amount of time.

In practice, we intend to continuously optimise the schedule. Firstly, we will use the list heuristic to generate a feasible solution. We then lock in a small segment of the initial solution, have the automated-system begin on the segment, while trying to improve the remaining schedule for the duration of that segment. If an improved schedule is obtained at the end of that segment is then the schedule that is followed. If new slides are added into the system, then we will cancel the optimisation process and instead incorporate the new slides into the existing schedule.

In this sense, we can adapt our approach from the offline version of the problem studied in this paper, to the online, dynamic real-world problem.

6.6 Model Extensions / Future Work

The problem considered in this chapter is an allocation and scheduling problem similar to those studied by Lombardi (2009). Our model avoids the allocation aspect of the problem by heuristically assigning staining projects to specific units. The benefit of this is that the exact travel times can be included in the weights of the precedence constraints and are thus known by the temporal network. Furthermore the location of the specific projects are used as parameters to the various sequence variables and state function that record the locations of the various robots. A limitation is that an optimal solution to the scheduling problem is not necessarily globally optimum when also considering the different units the projects can be allocated to. Future work could involve building this assignment component into the scheduling model. Furthermore it would be interesting to see if we could estimate how much that would be gained by doing so.

Another limitation to the model is that the Agitate and Micro clusters are all compulsory. Depending on the maximum time lags between consecutive agitates or micro scavenges, this can have a tightening effect on the temporal network. In reality, ideally the number of each of these tasks will depend on gap between the driving variables from clusters corresponding to the main actions of the current and next step in the staining protocol. This information is currently not considered by the model and its impact is not immediately clear.

Currently the model assumes that all dispenses of mixed chemical require the same quantity. This assumption was made such that the dummy clusters from the mixing protocols and the relevant dispense clusters from the staining protocol can be joined by an `isomorphism` constraint, which in our experience is by far the most effective method for communicating this information to the solver. If dispenses of mixed chemical required different quantity of the chemical, this would significantly complicate both how many dummy clusters per mixing protocol must be considered, as well as how the dummy and dispense clusters are merged. In our experience this assumption holds quite well in practice, however understanding the extent of this limitation is still not exactly clear.

Currently in this chapter we do not consider symmetry breaking. Staining projects even of the same class assigned to different units technically are not sym-

metric due to the travel time of the robots and gantry. However mixing projects of the same class are symmetric and this could be explored. From Chapter 3 of this thesis we would expect the symmetry breaking to potentially improve the lower bounds obtained, but the size of these instances we would not expect much significant performance improvement in terms of objective function.

There are a range of other heuristic solutions that could be run in parallel to the list heuristic. Firstly, as already discussed the slides could be allocated to different units. Secondly, the order in which projects are considered in the list heuristic could be permuted. Thirdly, instead of a list heuristic, it could be sufficient to simply consider the strongly-connected components of the temporal network in a specific order instead of the specific interval clusters. Fourthly, multiple staining projects could be fixed to different dummy clusters from the same mixing project. This should be beneficial to mixing projects with long expiry times such as DAB. Alternatively, it is possible to post-process the heuristic solutions in order to reduce the number of mixing projects required before passing this information to the full model.

6.7 Conclusion

In this chapter we modelled and solved the real-world scheduling problem that motivated this thesis using a framework based on the notion of interval clusters. This framework allows for the efficient modelling of complicated real-world scheduling problems that involve multiple levels of abstraction, and can still be solved efficiently. More specifically, it allows a scheduling problem to be modelled with a top level description of the driving variables and then a description of all the different types of interval clusters. For our real-world problem, this provided an intuitive way to describing, modelling, and solving the scheduling problem. Preliminary results motivate the further development of this approach and a range of possible extensions to the model have been proposed.

CHAPTER 7

Concluding Remarks

7.1 Contribution

This thesis is motivated by modelling and solving scheduling problems considered by fully-automated robotic systems that complete advanced cell staining: a process routinely used by pathologists to diagnose cancers and infectious diseases. By making efficient use of the resources of the system, it is possible to increase the system's throughput, while still adhering strictly to the specific staining protocols and thus ensuring the quality of the tests are maintained. Ultimately, the improved performance will allow pathologists to perform more tests in a given amount of time and therefore be more informed when making a diagnosis.

Although motivated by a specific application, the main aim of the thesis was to provide general insights and frameworks for understanding how related scheduling problems can be best modelled and solved by existing combinatorial solvers. To that end, the thesis makes the following novel contributions:

- **Symmetry Breaking -** Proposes and evaluates different methods of exploiting symmetry when considering scheduling problems with identical projects, specifically for the high-multiplicity RCPSP/max.
- **Driving Variables and LU Consistency -** Formalises a theory that allows one to reason about how assignments to specific subsets of variables can be extended to a complete solution at a fixed point. The method is based on a

novel local consistency measure, that we name LU consistency.

- **Interval Clusters -** Develops an intuitive framework based on the novel concept of interval clusters for modelling scheduling problems that reasons over multiple levels of abstraction. The framework builds upon the concepts of driving variables and LU consistency.

The thesis considers a range of scheduling problems that are considered for a number of purposes and relate in different ways to the real-world problem.

- **RCPSP/max -** We use the RCPSP/max as a reference problem when introducing the necessary terminology, concepts and solving technologies that were used throughout the thesis. We consider the RCPSP/max specifically as it is both well studied in the literature and, although not discussed explicitly in the thesis, can be used to model instances of the motivating scheduling problem following a series of conservative simplifying assumptions.
- **High-multiplicity RCPSP/max -** We consider instances of the RCPSP/max, referred to as high-multiplicity instances, that include multiple projects with identical characteristics. These arise naturally in instances of the real-world problem when the same staining protocol is used for multiple slides. In evaluating the different methods of symmetry breaking we were able to determine new best solutions to a number of instances in the well-establish dataset (MPSPlib). However, it was not immediately clear at this point how to add aspects of the real-world problem back into the high-multiplicity RCPSP/max and so we considered the LHRSP.
- **LHRSP -** To model and solve some complicated aspects of the motivating real-world problem, we proposed and studied the LHRSP. In modelling the problem with a CP Optimizer model, we proved that once a specific subset of the variables are fixed and constraint propagation reaches a fixed point, that it is always possible to obtain a complete solution through setting the remaining variables to specific values. We empirically demonstrated the significant improvement in performance that this insight offers.
- **LHRSP+ -** Upon formalising the concepts of driving variables and LU consistency, we considered a slightly more generalised version of the LHRSP that contained some more aspects of the real-world problem. Again we empirically demonstrated the significant improvement in performance that this insight offers.
- **Cell Staining Instrument Problem -** Building upon the work of the previous

chapters, in Chapter 6 we then modelled the real-world problem. The problem was not considered until Chapter 6 for a number of reasons. Firstly, as the robotic instrument has not yet been released to market there is sensitive information that we cannot yet make public. Secondly, we lacked a framework to describe, model and solve the problem in a simple way. The concept of interval clusters provided us this framework and the computational study demonstrated how good-quality, feasible solutions can be obtained quickly and then significantly improved.

In conclusion, the thesis contains both application-oriented and more fundamental research on effective modelling of scheduling problems. By focussing our research to understanding how to best leverage existing technologies we are able to develop theory that not only helps us with our real-world problem but also has more general applicability across different areas of scheduling and combinatorial optimisation. Testing these ideas on a wide range of problems is beyond the scope of this thesis.

7.2 Future Work

Directions of future work have been discussed at the end of each of the Chapters 3-6. Here we summarise the most important of these directions as well as make a number of new high-level directions arising from the work of the dissertation.

Formalising the concept of LU consistency is in our opinion the most salient contribution of the thesis. Not only did it allow for significantly improved performance of the solver, but, in association with driving variables and interval clusters, also provided the basis for which the motivating problem could eventually be modelled. From a methodological view-point, we believe practitioners solving problems with CP could significantly benefit from an understanding LU consistency. Hence future work should consider identifying existing problems in the literature where it is possible to demonstrate the power of LU consistency to highlight its more general applicability.

Currently LU consistency has only been proven to exist on special cases of global constraints primarily considering interval variables without a fixed length but with a compulsory region. A resource temporal network (Laborie, 2003b) can be seen as a generalisation of the global constraints considered. Hence relating LU consistency to resource temporal networks, and the seemingly the associated Necessary Truth Criterion, might help explain this connection. It is also interesting

to understand whether other existing global constraints are LU consistent for any useful partitions outside of the scheduling-oriented constraints considered in this thesis.

We believe the models in Chapters 5 and 6 can be improved. More explicitly, they contain some interval variables, for example aspirate and wash variables, that do not necessarily have a compulsory region when the driving variables are fixed. This can result in weak filtering while trying to feasibly assign the driving variables. As such, we believe that the models in both these chapters could be improved by removing such variables in a similar way that the models in Chapter 4 were improved by reducing the types of interval variables.

This thesis has largely built off the success of modern Constraint-Based Scheduling technologies. Given the similarities between scheduling and packing problems, where scheduling can be seen as a 1-dimensional version of either 2d or 3d packing, could Constraint-Based Packing be a future successful application domain for CP? We are aware of a number of global constraints specific to packing problems (Shaw et al., 2004) but have not seen any effort to create a packing specific modelling language on top of traditional CP in the way that CP Optimizer has done for scheduling. In such a modelling language, one could imagine generalising the notion of interval variables, from a 1 dimensional line, to higher dimensions such that they could represent 2d or 3d rectangles to pack. For such generalised variables, would the concept of LU consistency still be sufficient or would it also need to be generalised to, for example, Left-Right-Up-Down (LRUD) consistency for 2d packing problems?

In our opinion, the default search of CP Optimizer could be improved by considering the list heuristic used to obtain initial solutions in Chapter 6. The list heuristic is based on trying to insert the strongly connected components of the temporal network into schedule one at a time, similar to the direct method of Franck et al. (2001). Although this method does cannot guarantee that failures will not occur, the heuristic can efficiently generate feasible solutions with a small number of failures as shown in our results in Chapter 6. Futhermore, there is evidence to suggest that, particularly for some specific objective functions such as minimising total makespan, inserting activities by their components can often achieve superior results. In their winning approach to the MISTA 2013 scheduling competition, Asta et al. (2016) explicitly outline how they exploit this property.

We believe the symmetry breaking constraint considered in Chapter 3 could be extended to some special cases of multi-mode scheduling problems. The impact

on the solver performance of these symmetry breaking constraints were somewhat insignificant, particularly in terms of finding feasible solutions. However, from a theoretically perspective, understanding the extent that the symmetry can be generalised, and whether it can be detected efficiently, would still be worthwhile future research.

Finally, additional future work involves the further development and embedding of our scheduling approach into the robotic instrument that provided the motivation for this research. More explicitly, this consists of significantly more computational experiments to assess the performance of the approach under a wider range of situations such as online scenarios, as well as investigating whether the assignment of slides to processing units can be incorporated into the model.

APPENDIX A

Additional Material - Chapter 3

A.1 Tightening Constraints

In high-multiplicity scheduling problems it is common that the problems have a high resource utilisation, i.e., resources are required near their capacity for much of the project. A resource-based lower bound to any instance of the problem is the minimum amount of time such that one of the resources would need to be completely utilised to satisfy all the requirements. Constraints can be added to the MIP models to account for such lower bounds.

For the reduced model based on integer step and on/off variables, denoted ROOSDDT in Section 3.4.3, the following constraints are added,

$$\underbrace{R_k \sum_{\tau=t}^{T}(1 - z_{\omega,\tau})}_{\text{(a)}} \geq \underbrace{\sum_{c\in C}\sum_{i\in V_c}\left(d_{i,c}\cdot r_{i,c,k}\cdot(m_c - z_{i,c,t})\right.}_{\text{(b)}} + \underbrace{\left.\sum_{\tau=t^*}^{t}(t + d_{i,c} - \tau)\cdot(z_{i,c,\tau} - z_{i,c,\tau-1})\cdot r_{i,c,k}\right)}_{\text{(c)}} \quad \forall\, k \in \mathcal{R}; t \in H \tag{A.1}$$

where $t^* = \max(0, t - d_{i,c})$. Term (a) represents the total amount of energy that

is available over the interval from t until the dummy end variable S_ω. Term (b) represents the amount of energy required by all activities that have not started by time t. Term (c) represents the total amount of energy required after time t by the subset of activities that have started processing before or at t. Hence these constraints ensure that the start of the dummy-end activity, and hence the length of the total project, is long enough such that there is sufficient amount of resource to process all of the activities.

The resource-based tightening constraints for the other models considered can be expressed similarly. For the reduced formulation based on integer pulse start variables, denoted RDDT in Section 3.4.2, (using $z_{i,c,t} = \sum_{\tau=0}^{t} y_{i,c,\tau}$), and for the corresponding extended version of the model, denoted DDT in Section 3.4.1, the resource-based constraints are given in (A.2) and (A.3) respectively:

$$R_k \left(T - t - \sum_{\tau=t}^{T} \sum_{\tau'=0}^{\tau} y_{\omega,\tau'} \right) \geq \sum_{c \in C} \sum_{i \in V_c} \left(d_{i,c} r_{i,c,k} \cdot (m_c - \sum_{\tau=0}^{t} y_{i,c,\tau}) + \sum_{\tau=t^*}^{t} (t + d_{i,c} - \tau)\, y_{i,c,\tau}\, r_{i,c,k} \right) \quad \forall\, k \in \mathcal{R};\ t \in H \tag{A.2}$$

$$R_k \left(T - t - \sum_{\tau=t}^{T} \sum_{\tau'=0}^{\tau} x_{\omega,\tau'} \right) \geq \sum_{(i,p,c) \in V} \left(d_{i,c} \cdot r_{i,c,k} (1 - \sum_{\tau=0}^{t} x_{i,p,c,\tau}) + \sum_{\tau=t^*}^{t} (t + d_{i,c} - \tau) x_{i,p,c,\tau} r_{i,c,k} \right) \quad \forall\, k \in \mathcal{R};\ t \in H \tag{A.3}$$

Constraints (A.3) can be thought of a special case of constraints (A.2) where all projects belonged to their own class.

A.2 Polyhedral Analysis

Proposition A.2.1. $RDDT \prec^{LP} DDT$

Proof. Need a transformation of DDT on the solution space P(RDDT) that gives formulation DDT′ and solution space P(DDT′) such that $P(DDT') \subseteq P(RDDT)$. Let $\overline{y}_{i,c,t} = \sum_{p \in P_c} \overline{x}_{i,p,c,t}$ be the orthogonal projection (thus affine transformation) of DDT onto the solution space of RDDT.

Recall the set of all precedences was defined as $A := A^{real} \cup A^{start} \cup A^{end}$,

where A^{start} are the precedences from the dummy start node of the master projects to the dummy start nodes of the individual project, A^{end} are the precedences from the dummy end nodes of the individual projects to the master dummy end node, and A^{real} are the precedences within the individual projects. As the objective function considered is to minimise makespan it is possible to neglect the A^{start} precedences as we can assume that start times of the dummy start activities for individual projects will start as early as possible. Hence it is possible to consider the precedence constraints (3.5b) in DDT as the following two constraints,

$$\sum_{\tau=0}^{t-\delta_{(i,i',c)}} x_{i,p,c,\tau} - \sum_{\tau=0}^{t} x_{i',p,c,\tau} \geq 0 \qquad ((i,i',c) \in A_c; p \in P_c; c \in C; t \in H) \quad \text{(3.5b-a)}$$

$$\sum_{\tau=0}^{t} x_{\omega,p,c,\tau} - \sum_{\tau=0}^{t} x_{\omega,\tau} \geq 0 \qquad (p \in P_c; c \in C; t \in H) \quad \text{(3.5b-b)}$$

Here constraints (3.5b-a) represent the precedences from A^{real} and constraints (3.5b-b) represent the precedences from A^{end}. Now by considering the affine transformation for constraints (3.5b-a), we obtain the following,

$$\sum_{p \in P_c} \Big(\sum_{\tau=0}^{t-\delta_{(i,i',c)}} \overline{x}_{i,p,c,\tau} - \sum_{\tau=0}^{t} \overline{x}_{i',p,c,\tau} \Big) \geq 0 \qquad ((i,i',c) \in A_c; c \in C; t \in H)$$

$$\sum_{\tau=0}^{t-\delta_{(i,i',c)}} \overline{y}_{i,p,c,\tau} - \sum_{\tau=0}^{t} \overline{y}_{i',p,c,\tau} \geq 0 \qquad ((i,i',c) \in A_c; c \in C; t \in H)$$

Which are equivalent to constraints (3.6b) from the RDDT. A similar argument can be made to show that after the transformation the resource constraints (3.5c) and (3.6c) are equivalent, constraints (3.5d) and (3.6d) are equivalents, and constraints (3.5f) and (3.6g) are equivalent. It remains to show that, after the affine transformation, the constraints (3.5b-b) are not weaker than either constraints (3.6e) or (3.6f).

It is possible to show that this is true for constraints (3.6e) as follows,

$$0 \leq \sum_{\tau=0}^{t} \overline{x}_{\omega,p,c,\tau} - \sum_{\tau=0}^{t} \overline{x}_{\omega,\tau} \qquad (c \in C; p \in P_c; t \in H)$$

$$\leq \sum_{p \in P_c} (\sum_{\tau=0}^{t} \overline{x}_{\omega,p,c,\tau} - \sum_{\tau=0}^{t} \overline{x}_{\omega,\tau}) \qquad (c \in C; t \in H)$$

$$= \sum_{p \in P_c} \sum_{\tau=0}^{t} \overline{x}_{\omega,p,c,\tau} - m_c \sum_{\tau=0}^{t} \overline{x}_{\omega,\tau} \qquad (c \in C; t \in H)$$

$$= \sum_{\tau=0}^{t} \overline{y}_{\omega,p,c,\tau} - m_c \sum_{\tau=0}^{t} \overline{y}_{\omega,\tau} \qquad (c \in C; t \in H)$$

Let us now consider (3.6f) as follows,

Firstly recall that $X_c := \{i \in V_c | (i, \omega) \in A_c \wedge \mu_{i,c} < m_c\}$ represent the set of activities for class $c \in C$ that directly precede the dummy end node of the individual projects of the class and that due to resource requirements all required multiples cannot overlap in time.

$$\sum_{\tau=0}^{t-d_{i,c}} \overline{x}_{i,p,c,\tau} \geq \sum_{\tau=0}^{t} \overline{x}_{\omega,p,c,\tau} \geq \sum_{\tau=0}^{t} \overline{x}_{\omega,\tau} \qquad (i \in X_c; c \in C; t \in T)$$

By instead summing over the intervals after time unit t as follows,

$$(1 - \sum_{\tau=t-d_{i,c}}^{T} \overline{x}_{i,p,c,\tau}) \geq (1 - \sum_{\tau=t}^{T} \overline{x}_{\omega,\tau}) \qquad (i \in X_c; c \in C; t \in T) \qquad (\star)$$

It is possible to rearrange these constraints to more closely resemble constraints (3.6f) as follows.

$$\sum_{\tau=t}^{T} \overline{x}_{\omega,\tau} \geq \sum_{\tau=t-d_{i,c}}^{T} \overline{x}_{i,p,c,\tau} \qquad (i \in X_c; c \in C; t \in T)$$

$$\geq \overline{x}_{i,p,c,t-d_{i,c}} \qquad (i \in X_c; c \in C; t \in T)$$

$$\geq \frac{1}{\mu_{i,c}} \sum_{p \in P_c} \overline{x}_{i,p,c,t-d_{i,c}} \qquad (i \in X_c; c \in C; t \in T)$$

The first inequality comes directly from rearranging $(\star)$. The second inequality is clearly true as $t - d_{i,c} \in [t - d_{i,c}, T[$. The third inequality is true due to the definition of $\mu_{i,c}$. Now considering the affine transformation and rearranging we obtain,

$$\overline{y}_{i,c,t-d_{i,c}} \leq \mu_{i,c} \sum_{\tau=t}^{T} \overline{y}_{\omega,\tau} \qquad (i \in X_c, c \in C, t \in H)$$

Which are clearly equivalent to constraints (3.6f).

To complete the proof we can provide a solution to the linear relaxation of RDDT that is infeasible in DDT. Consider the simple instance with a single class c containing a single activity i with a multiplicity $m_c = 3$ and duration $d_i = 1$ with no precedence constraints and no resource constraints.

The LP relaxation of (RDDT), we obtain

$$\sum_{\tau=0}^{t-1} y_{i,c,\tau} - \sum_{\tau=0}^{t} y_{\omega,c,\tau} \geq 0 \qquad (t = 1, 2)$$

$$\sum_{\tau=0}^{t} y_{\omega,\tau} - 3 \sum_{\tau=0}^{t} y_{\omega,c,\tau} \geq 0 \qquad (t = 1, 2)$$

$$\sum_{t \in [0,1,2]} y_{i,c,t} = 3, \quad \sum_{t \in [0,1,2]} y_{\omega,c,t} = 3, \quad \sum_{t \in [0,1,2]} y_{\omega,t} = 1,$$

$$y_{\omega,c,t-1} \leq 2 \sum_{\tau=t}^{3} y_{\omega,\tau}$$

$$0 \leq y_{i,c,t} \leq 2 \qquad (t = 0, 1, 2)$$

A solution to the above linear relaxation, which turns out to be optimal, is given in the table below

t	0	1	2
$y_{i,c,t}$	2	1	0
$y_{\omega,c,t}$	0	2	1
$y_{\omega,t}$	0	2/3	1/3

It is possible to show that the corresponding solution in DDT would violate the disaggregated precedence constraints (3.5b). As $y_{\omega,c,1} = 1$ there must exist a project

$p \in P_c$ such that $x_{\omega,p,c,1} = 1$. Now considering the disaggregated precedence constraint between (ω, p, c) and the master dummy end node (ω, c) we can see that for $t = 1$, the constraint

$$\underbrace{x_{\omega,p,c,0}}_{0} + \underbrace{x_{\omega,p,c,1}}_{0} - \underbrace{x_{\omega,0}}_{0} - \underbrace{x_{\omega,1}}_{2/3} \geq 0$$

is violated as $0 - 2/3 \ngeq 0$. □

Proposition A.2.2. *DDT $\prec^{LP}$ DDT-Sym*

Proof. Clearly DDT $\preceq^{LP}$ DDT-Sym as DDT-Sym was obtained from DDT by considering the additional precedence constraints A_1^{sym} and A_2^{sym}, and thus clearly any solution to DDT-Sym is also a solution to DDT. To complete the proof consider the simple instance with a single resource k with capacity $R_k = 1$, a single class c with a multiplicity $m_c = 2$, containing a single activity i with duration $d_i = 1$, and resource requirement $r_{i,c,k} = 1$. The time-horizon is 3.

The LP relaxation of (DDT), we obtain

$$\sum_{\tau=0}^{t-1} x_{i,p,c,\tau} - \sum_{\tau=0}^{t} x_{\omega,p,c,\tau} \geq 0 \qquad (p = 0,1; t = 1,2)$$

$$\sum_{\tau=0}^{t} x_{\omega,p,c,\tau} - \sum_{\tau=0}^{t} x_{\omega,\tau} \geq 0 \qquad (p = 0,1; t = 1,2)$$

$$\sum_{t\in[0,1,2]} x_{i,p,c,t} = 1, \sum_{t\in[0,1,2]} x_{\omega,c,t} = 1, \sum_{t\in[0,1,2]} x_{\omega,t} = 1,$$

$$\sum_{p\in[1,2]} x_{i,p,c,t} \leq 1, \qquad (t = 0,1,2)$$

$$0 \leq x_{i,p,c,t}, x_{\omega,p,c,t} \leq 1 \qquad (p = 0,1; t = 0,1,2)$$

$$0 \leq x_{\omega,t} \leq 1 \qquad (t = 0,1,2)$$

A solution to the above linear relaxation, which turns out to be optimal, is given in the table below

t	0	1	2
$x_{i,p,c,t}$	0.5	0.5	0
$x_{\omega,p,c,t}$	0	0.5	0.5
$x_{\omega,t}$	0	0.5	0.5

The LP relaxation of (DDT-Sym) can be obtained by considering the additional constraints.

$$\sum_{\tau=0}^{t-1} x_{i,0,c,\tau} - \sum_{\tau=0}^{t} x_{i,1,c,\tau} \geq 0 \qquad (t = 1, 2)$$

These constraints come from the strengthening constraints A_2^{sym}. Considering now the solution proposed above on the constraint for $t = 1$, i.e.,

$$\underbrace{x_{i,0,c,0}}_{0.5} - \underbrace{x_{i,1,c,0}}_{0.5} - \underbrace{x_{i,1,c,1}}_{0.5} \geq 0$$

Thus $-0.5 \ngeq 0$ and hence the solution violates the constraint. □

As we have modelled the ROOSDDT and RDDT explicitly we will now consider whether the reduced version of the models are polyhedral equivalent up to linear transformation.

Proposition A.2.3. *ROOSDDT* $\equiv^{LP}$ *RDDT*

Proof. By substituting constraints (3.7c) into (3.7d), the resource constraints of ROOSDDT can be expressed as

$$\sum_{\psi \in \Psi} r_{\psi,k} \sum_{(i,p,c) \in \overline{V}_\psi} (z_{i,c,t} - z_{i,c,t-d_{i,c}}) \leq R_k \quad (t \in H; k \in \mathcal{R}) \tag{A.4}$$

or equivalently

$$\sum_{c \in C} \sum_{i \in V_c} r_{i,c,k} (z_{i,c,t} - z_{i,c,t-d_{i,c}}) \leq R_k \quad (t \in H; k \in \mathcal{R}) \tag{A.5}$$

Hence ROOSDDT can be expressed entirely in terms of the integer-variable version of the step constraints. The integer step variables can then be transformed to the integer pulse variables by the following transformation, $z_{i,c,t} = \sum_{\tau=0}^{t} y_{i,c,\tau}$. Conversely the inverse transformation defines $y_{i,c,t} = z_{i,c,t} - z_{i,c,t-1}$ gives the step-variable only version from the ROOSDDT from the RDDT. □

Proposition A.2.4. *Let DDT+ be the DDT model with the resource-based tightening constraints (A.3).* $DDT \prec^{LP} DDT+$

Proof. Clearly $DDT \preceq^{LP} DDT+$ as $DDT+$ is constructed by adding additional

constraints to DDT. Hence it remains to show that there exists a solution to DDT that is not in the solution space of $DDT+$. We consider the same instance used in the proof to Proposition A.2.2. Hence we obtain the same relaxation to (DDT) as given in that proof. The relaxation to (DDT+) can be obtained by considering the additional tightening constraints, consider the constraint that corresponds to the time index $t = 0$ below,

$$2 - x_{\omega,1} - 2x_{\omega,0} \geq \sum_{p\in[1,2]} ((1 - x_{i,p,c,0}) + x_{i,p,c,0})$$

or equivalently

$$2 - \underbrace{x_{\omega,1}}_{0.5} - 2\underbrace{x_{\omega,0}}_{0} \geq \sum_{p\in[1,2]} (1) = 2$$

where again we use the solution to DDT used in the proof of Proposition A.2.2. Thus $1.5 \ngeq 2$ and hence the solution violates at least one of the additional constraints. □

Proposition A.2.5. *For the total-makespan objective*

$$RDDT \equiv^{LP} ROOSDDT \prec^{LP}$$
$$DDT \equiv^{LP} OOSDDT \prec^{LP}$$
$$DDT\text{-}Sym \equiv^{LP} OOSDDT\text{-}Sym$$

Proof. From Proposition A.2.1 we have $RDDT \prec^{LP} DDT$. From Proposition A.2.2, we have $DDT \prec^{LP}$ DDT-Sym. From Proposition A.2.3 we have $ROOSDDT \equiv^{LP} RDDT$. The model for $OOSDDT$ is not given explicitly in the paper, instead it is constructed from $ROOSDDT$ by redefining the set of project classes such that all projects are defined to belong to their own class. This is the same method in which DDT can be constructed from $RDDT$. Hence as we know that $RDDT \equiv^{LP} ROOSDDT$ and we know that both DDT and OOSDDT can be constructed in the same way by redefining sets to RDDT and ROOSDDT respectively, then we have DDT $\equiv^{LP}$ OOSDDT. The same logic holds for DDT-Sym $\equiv^{LP}$ OOSDDT-Sym, as the models are constructed from DDT and OOSDDT respectively by considering the same additional set of precedence constraints. □

APPENDIX B

Additional Material - Chapter 4

B.1 MIP Model for LHRSP

In the mathematical programming approach, the problem is modelled as a routing problem by the use of a directed, weighted multi-graph $G(V, A)$. Each operation $(i, j) \in O$ is assigned a vertex in the graph, $v(i, j)$, as well as a dummy start and dummy end node, denoted by 0 and γ respectively. Hence $V := \{0\} \cup \{v(i, j) | i, j \in O\} \cup \{\gamma\}$. In a multi-graph, there may be multiple arcs from one node to another, referred to as different *routes*. Hence we use triplet notation to represent arcs, where $(v, v', r) \in A$ represents an arc from node v to v' along route r and has weight $w_{v,v',r}$. The arc set is the union of two disjoint subsets $A = A_1 \cup A_2$, each of which contain at most a single arc from one node to another. Hence there is at most two routes from one node to another in the multi-graph G.

Arc set A_1 are referred to as the *slow routes*. An arc $(v, v', 1) \in A_1$ has a weight that represents the minimum amount of time required between the start of $v \in V$ and start of $v' \in V$ if the robot must travel to and from the vials to aspirate chemicals in between the two operations. More explicitly the arc set and equivalent weights can be defined as

- An arc from the dummy start to the node corresponding to the first operation of all jobs, i.e., $(0, v(1, j), 1) \in A_1$, with arc weight $w_{0,v(1,j),1} = p^{\uparrow} + p^{\rightarrow}_{0,j}$, for all $j \in J$. The arc weight comes from the robot aspirating chemical and then

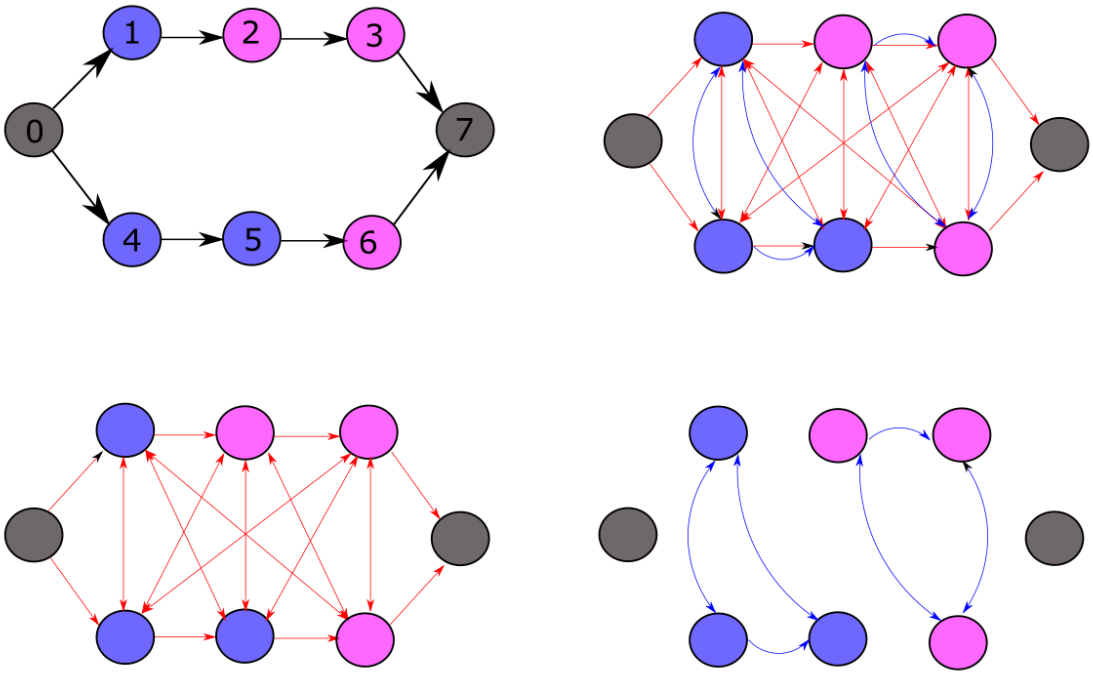


Figure B.1: An example of how the multigraph is constructed from imposing two networks. The top-left graph shows the precedence graph of two jobs with three operations each, where the colour of the nodes indicates which of the two chemicals are required. The top-right graph shows the multi-graph that is used in the routing problem, which comes from superimposing the slow routes (bottom-left graph) and fast routes (bottom-right graph)

travelling to the correct location.

- An arc between operations from different jobs or consecutive operations of the same job, i.e., $(v(i,j), v(i',j'), 1) \in A_1$ for all $(i,j), (i',j') \in O$ given that $j \neq j'$ or $(i',j') = (i+1, j)$ with weight $w_{v(i,j),v(i',j'),1} = p^{\downarrow} + p^{\rightarrow}_{j,0} + p^{\uparrow} + p^{\rightarrow}_{0,j'}$. The arc weight comes from the robot dispensing chemical at job j, moving to the vials, aspirating the next appropriate chemical, and then travelling to job j'.
- An arc from the last operation of all jobs to the dummy end, i.e., $(v(N_j, j), \gamma, 1) \in A_1$ for all $j \in J$ with arc weight $w_{v(N_j,j),\gamma,1} = p^{\downarrow}$. The arc weight is simply the time required to dispense the chemical.

Arc set A_2 can be thought of as the *fast routes*. An arc $(v, v') \in A_2$ has a weight that represents the minimum amount of time required between the start of v and v' if the robot travels directly between the two without aspirating new chemical in between. Since the robot can not carry different chemicals at the same time, arcs can only exist in A_2 if both the predecessor and successor require the same chemical.

More explicitly, A_2 and the associated weights can be defined as

- An arc between operations that require the same chemical from different jobs or consecutive operations of the same jobs, i.e., $(v(i,j), v(i',j'), 2)$ for all $(i,j),(i',j') \in O^c$ given that $j \neq j'$ or $(i',j') = (i+1,j)$ with weight $p^{\downarrow} + p^{\rightarrow}_{j,j'}$. The arc weight comes from the robot dispensing chemical $c \in C$ at job j and then travelling directly from job j to j' to dispense more of chemical c.

A small example illustrating how the multigraph is constructed is shown in Figure B.1. The example considers an instance with two jobs, each of which contain three operations, and two colours. The precedence graph is also shown, where evidently nodes $\{1,2,3\}$ belong to one job and $\{4,5,6\}$ belong to the other. Furthermore nodes 0 and 7 are the dummy start and end nodes respectively.

In order to tighten the mathematical model, for each vertex $v \in V$ an earliest start time ES_v and latest start time LS_v are defined. The ES_v can be calculated by the longest path between 0 and v in the precedence graph. Whereas LS_v can be set to M minus the length of the longest path between v and γ, where M is a sufficiently large number that we generally obtain from from a heuristic. Finally for ease of notation let $V^c := \{v(i,j) | (i,j) \in O^c\}$ be the set of nodes associated with operations requiring chemical $c \in C$.

The entire mathematical model can now be formulated as,

$$\text{Min.} \quad S_\gamma \tag{B.1a}$$

$$\text{s.t.} \sum_{(v',v,r)\in\delta^-(v)} x_{v',v,r} = 1, \qquad (v \in V) \tag{B.1b}$$

$$\sum_{(v,v',r)\in\delta^+(v)} x_{v,v',r} = 1 \qquad (v \in V) \tag{B.1c}$$

$$\sum_{\substack{((v,v',r)\in A_2:\\ v\in P \wedge v'\in P)}} x_{v,v',r} \leq L_c - 1, \qquad (c \in C, P \subseteq V^c : |P| = L_c + 1) \tag{B.1d}$$

$$\ell^{min}_{i,j} \leq S_{v(i+1,j)} - S_{v(i,j)} \leq \ell^{max}_{i,j}, \qquad (i \in O_j \setminus \{N_j\}; j \in J) \tag{B.1e}$$

$$S_v - M_{v,v'}(1 - x_{v,v',r}) \leq \tag{B.1f}$$

$$S'_v - \sum_{\substack{r'\in\{1,2\}:\\ (v,v',r')\in A}} w_{v,v',r'} x_{v,v',r'} \qquad ((v,v',r) \in A | v \neq \gamma)$$

$$\sum_{(v,v',r)\in A} w_{v,v',r} x_{v,v',r} \leq S_\gamma \tag{B.1g}$$

$$\sum_{v \in V_c} \sum_{\substack{(v,v',r) \in \delta^+(v): \\ r=2}} x_{v,v',r} \geq \bar{\alpha}_c, \qquad (c \in C) \quad \text{(B.1h)}$$

$$x_{v,v',r} \in \{0,1\} \qquad ((v,v',r) \in A) \quad \text{(B.1i)}$$

$$ES_v \leq S_v \leq LS_v \qquad (v \in V) \quad \text{(B.1j)}$$

Binary variables $x_{v,v',r}$ denote whether the robot moves from v to v' along route r. Continuous variable S_v records the time that the robot arrives at node v. In addition, S_γ records the makespan of the schedule.

The objective is to minimise the makespan, or equivalently, the start time of the dummy variable, specified in (B.1a). Constraints (B.1b) and (B.1c) enforce that the in-degree and out-degree of each node is exactly one, respectively, where $\delta^-(v)$ and $\delta^+(v)$ represent the incoming and outgoing arcs from node $v \in V$ respectively. Constraint (B.1d) is a subpath elimination constraint, which ensures that the robot does not dispense more chemical than it can carry. Given the exponential number of the subpath elimination constraints, these are added lazily into the model at each integer solution. At each integer solution we can determine the sets of nodes between which the robot travels only using the fast routes, $\overline{P} := \{\overline{P}_1, \overline{P}_2, ...\}$. Given the structure of the multigraph, each of these sets of nodes, $\overline{P}_p \in \overline{P}$ are associated with a specific colour, $c(\overline{P}_p) \in C$. If the size of this set of nodes exceeds the possible limit, $|\overline{P}_p| > L_{c(P_p)}$, we add lazy constraints for each subset $P \subseteq \overline{P}_p$ of the sets of nodes according to (B.1d).

Constraints (B.1e) enforce the minimum and maximum time-lags between adjacent operations from the same job. Constraints (B.1f) are linking constraints that relate the arrival time of a node to the arrival time of the node that the robot directly travels from, here $M_{v,v'} := LS_v - ES'_v$ is a sufficiently large number. Moreover, these linking constraints remove the need for sub-tour elimination constraints that are required by similar MIP formulations for related routing problems. It should be noted that in constraints (B.1f) if there exists two arcs between the considered nodes then we sum the weights of both arcs to tighten fractional solutions. Constraints (B.1g) and (B.1h) are tightening constraints. Constraint (B.1g) enforce that the makespan is at least larger than the weight of all the arcs completed. Whereas constraints (B.1h) ensure that the number of slow routes exiting nodes requiring chemical $c \in C$ is at least equal to the minimum number of aspirates, α_c, which can be calculated by $\bar{\alpha}_c = \left\lceil \frac{|O^c|}{L_c} \right\rceil$. Finally constraints (B.1i) state that the $x_{v,v',k}$ are binary and constraints (B.1j) are continuous within the range of the earliest and latest start times.

Bibliography

Achterberg, T. *Constraint Integer Programming*. PhD thesis, Technical University of Berlin, 2007.

Aggoun, A. and Beldiceanu, N. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993. doi: 10.1016/0895-7177(93)90068-A.

Alvarez-Valdés, R. and Tamarit, J. The project scheduling polyhedron: Dimension, facets and lifting theorems. *European Journal of Operational Research*, 67(2):204–220, 1993. doi: 10.1016/0377-2217(93)90062-R.

Arnborg, S., Corneil, D., and Proskurowski, A. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

Artigues, C. On the strength of time-indexed formulations for the resource-constrained project scheduling problem. *Operations Research Letters*, 45(2):154–159, 2017.

Artigues, C., Michelon, P., and Reusser, S. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149(2):249–267, 2003. doi: 10.1016/S0377-2217(02)00758-0.

Artigues, C., Huguet, M.-J., and Lopez, P. Generalized disjunctive constraint propagation for solving the job shop problem with time lags. *Engineering Applications of Artificial Intelligence*, 24(2):220–231, 2011. doi: 10.1016/j.engappai.2010.07.008.

Artigues, C., Koné, O., Lopez, P., and Mongeau, M. *Mixed-Integer Linear Programming Formulations*, pages 17–41. Springer International Publishing, Cham, 2015. doi: 10.1007/978-3-319-05443-8_2.

Asta, S., Karapetyan, D., Kheiri, A., Özcan, E., and Parkes, A. J. Combining Monte-Carlo and hyper-heuristic methods for the problem. *Information Sciences*, 373:476–498, 2016. doi: 10.1016/j.ins.2016.09.010.

Baatar, D., Krishnamoorthy, M., and Ernst, A. T. A Triplet-Based Exact Method for the Shift Minimisation Personnel Task Scheduling Problem. In Bansal, N. and Finocchi, I., editors, *Algorithms - ESA 2015*, pages 59–70, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

Bánkfalvi, A., Boecker, W., and Reiner, A. Comparison of automated and manual determination of HER2 status in breast cancer for diagnostic use: a comparative methodological study using the Ventana BenchMark automated staining system and manual tests. *International Journal of Oncology*, 25(4):929–964, 2004. doi: https://doi.org/10.3892/ijo.25.4.929.

Bartusch, M., Möhring, R. H., and Radermacher, F. J. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16(1):199–240, 1988. doi: 10.1007/BF02283745.

Beck, C. J., Prosser, P., and Selensky, E. Vehicle Routing and Job Shop Scheduling : What's the difference? *13th International Conference on Artificial Intelligence Planning and Scheduling (ICAPS)*, pages 267–276, 2003.

Bessiere, C. Constraint Propagation. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 29–83. Elsevier, 2006. doi: 10.1016/S1574-6526(06)80007-6.

Bianco, L. and Caramia, M. A new formulation for the project scheduling problem under limited resources. *Flexible Services and Manufacturing Journal*, 25(1-2):6–24, 2013. doi: 10.1007/s10696-011-9127-y.

Biosystems, L. Leica ChromoPlex^{TM} 1 Dual Detection for BOND Fully Automated 1-Step Parallel Staining, 2012.

Biosystems, L. BOND-MAX Fully automated IHC and ISH, 2019.

Blazewicz, J., Lenstra, J., and Kan, A. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983. doi: 10.1016/0166-218X(83)90012-4.

Bodlaender, H. L. A linear time algorithm for finding tree-decompositions of small treewidth. In *STOC '93 Proceedings of the twenty-fifth annual ACM symposium on Theory of Computing*, pages 226–234, 1993.

Booth, K. E., Tran, T. T., Nejat, G., and Beck, C. J. Mixed-Integer and Constraint Programming Techniques for Mobile Robot Task Planning. *IEEE Robotics and Automation Letters*, 1(1):500–507, 2016. doi: 10.1109/LRA.2016.2522096.

Bordeaux, L., Katsirelos, G., Narodytska, N., and Vardi, M. Y. The complexity of integer bound propagation. *Journal of Artificial Intelligence Research*, 40:657–676, 2011. doi: 10.1613/jair.3248.

Boudoukh, T., Penn, M., and Weiss, G. Scheduling jobshops with some identical or similar jobs. *Journal of Scheduling*, 4(4):177–199, 2001. doi: 10.1002/jos.72.

Boussemart, F., Hemery, F., Lecoutre, C., and Sais, L. Boosting Systematic Search by Weighting Constraints. In *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, pages 146–150, Amsterdam, The Netherlands, The Netherlands, 2004. IOS Press.

Brafman, R. I. A simplifier for propositional formulas with many binary clauses. *IJCAI International Joint Conference on Artificial Intelligence*, 34(1):515–520, 2001. doi: 10.1109/TSMCB.2002.805807.

Brinkmann, K. and Neumann, K. Heuristic procedures for resource—constrained project scheduling with minimal and maximal time lags: the resource—levelling and minimum project—duration problems. *Journal of Decision Systems*, 5(1-2):129–155, 1996. doi: 10.1080/12460125.1996.10511678.

Brucker, P., Drexl, A., Mohring, R., Neumann, K., and Pesch, E. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112 (1):3–41, 1999a. doi: 10.1016/S0377-2217(98)00204-5.

Brucker, P., Hilbig, T., and Hurink, J. A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics*, 94(1-3):77–99, 1999b. doi: 10.1016/S0166-218X(99)00015-3.

Caumond, A., Lacomme, P., and Tchernev, N. A memetic algorithm for the job-shop with time-lags. *Computers and Operations Research*, 35(7):2331–2356, 2008. doi: 10.1016/j.cor.2006.11.007.

Chand, S. *Automated Design of Heuristics for the Resource Constrained Project Scheduling Problem*. PhD thesis, The University of New South Wales, 2018.

Chand, S., Huynh, Q., Singh, H., Ray, T., and Wagner, M. On the use of genetic programming to evolve priority rules for resource constrained project scheduling problems. *Information Sciences*, 432:146–163, 2018. doi: 10.1016/j.ins.2017.12.013.

Cheng, B. M. W., Choi, K. M. F., Lee, J. H. M., and Wu, J. C. K. Increasing Constraint Propagation by Redundant Modeling: an Experience Report. *Constraints*, 4(2):167–192, may 1999. doi: 10.1023/A:1009894810205.

Christofides, N., Alvarez-Valdes, R., and Tamarit, J. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29(3):262–273, 1987. doi: 10.1016/0377-2217(87)90240-2.

Chu, G. *Improving Combinatorial Optimization*. PhD thesis, The University of Melbourne, 2011.

Confessore, G., Giordani, S., and Rismondo, S. A market-based multi-agent system model for decentralized multi-project scheduling. *Annals of Operations Research*, 150(1):115–135, 2007. doi: 10.1007/s10479-006-0158-9.

Cook, S. A. The complexity of theorem-proving procedures. In *STOC '71 Proceedings of the third annual ACM symposium on Theory of computing*, volume 15, pages 151–158, Cook1971, 1971. doi: 10.1145/800157.805047.

Dechter, R. chapter 2 - Constraint Networks. In Dechter, R., editor, *Constraint Processing*, The Morgan Kaufmann Series in Artificial Intelligence, pages 25–49. Morgan Kaufmann, San Francisco, 2003a. doi: https://doi.org/10.1016/B978-155860890-0/50003-7.

Dechter, R. chapter 4 - Directional Consistency. In Dechter, R., editor, *Constraint Processing*, The Morgan Kaufmann Series in Artificial Intelligence, pages 85–115. Morgan Kaufmann, San Francisco, 2003b. doi: https://doi.org/10.1016/B978-155860890-0/50005-0.

Dechter, R. chapter 8 - Advanced Consistency Methods. In Dechter, R., editor, *Constraint Processing*, The Morgan Kaufmann Series in Artificial Intelligence, pages 211–243. Morgan Kaufmann, San Francisco, 2003c. doi: https://doi.org/10.1016/B978-155860890-0/50009-8.

Dechter, R. Chapter 7 - Tractable Structures for Constraint Satisfaction Problems. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 209–244. Elsevier, 2006. doi: https://doi.org/10.1016/S1574-6526(06)80011-8.

Dohn, A., Rasmussen, M. S., and Larsen, J. The vehicle routing problem with time windows and temporal dependencies. *Networks*, 58(4):273–289, dec 2011. doi: 10.1002/net.20472.

Drexl, A., Nissen, R., Patterson, J. H., and Salewski, F. ProGen/$\pi\chi$ - an instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions. *European Journal of Operational Research*, 125(1):59–72, 2000. doi: 10.1016/S0377-2217(99)00205-2.

Du, J., Leung, J. Y.-T., and Young, G. H. Scheduling chain-structured tasks to minimize makespan and mean flow time. *Information and Computation*, 92(2):219–236, 1991. doi: 10.1016/0890-5401(91)90009-Q.

Edwards, S. J., Baatar, D., Bowly, S., and Smith-Miles, K. Symmetry breaking in a special case of the RCPSP / max. *Proceedings of the 8th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2017), 05 - 08 Dec 2017, Kuala Lumpur, Malaysia*, pages 315–318, 2017.

Edwards, S. J., Baatar, D., Ernst, A. T., and Smith-Miles, K. The Liquid Handling Robot Scheduling Problem. In *Scheduling and Planning Application woRKshop (SPARK'18) in ICAPS*, pages 18–26,

Delft, 2018.

Edwards, S. J., Baatar, D., Smith-Miles, K., and Ernst, A. T. Symmetry breaking of identical projects in the high-multiplicity RCPSP/max. *Journal of the Operational Research Society*, 0(0):1–22, 2019. doi: 10.1080/01605682.2019.1595192.

Erschler, J. and Lopez, P. Energy-based approach for task scheduling under time and resources constraints. In *2nd International Workshop on Project Management and Scheduling*, pages 115–121, 1990.

Fahimi, H., Ouellet, Y., and Quimper, C.-g. Linear-time filtering algorithms for the disjunctive constraint and a quadratic filtering algorithm for the cumulative not-first not-last. *Constraints*, 23(3):272–293, 2018.

Feydy, T. *Constraint programming: improving propagation*. PhD thesis, University of Melbourne, Department of Computer Science and Software Engineering, 2010.

Feydy, T., Somogyi, Z., and Stuckey, P. J. Half Reification and Flattening. In Lee, J., editor, *Principles and Practice of Constraint Programming*, pages 286–301. Springer, 2011.

Franck, B., Neumann, K., and Schwindt, C. Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR-Spektrum*, 23(3):297–324, 2001. doi: 10.1007/PL00013356.

Frank, J., Do, M., and Tran, T. T. Scheduling Ocean Color Observations for a GEO-Stationary Satellite. (Icaps):376–384, 2016.

Freuder, E. In pursuit of the holy grail. *Constraints*, 2:57–61, 1997. doi: 10.1145/242224.242304.

Freuder, E. C. A Sufficient Condition for Backtrack-Free Search. *Journal of the Association for Computing Machinery*, 29(I):24–32, 1982.

Gaspers, S., Misra, N., Ordyniak, S., Szeider, S., and Živný, S. Backdoors into heterogeneous classes of SAT and CSP. *Journal of Computer and System Sciences*, 85:38–56, 2017a. doi: 10.1016/j.jcss.2016.10.007.

Gaspers, S., Ordyniak, S., and Szeider, S. Backdoor Sets for CSP. In *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7, pages 137–157. Dagstuhl Publishing, 2017b.

Geelen, P. A. Dual Viewpoint Heuristics for Binary Constraint Satisfaction Problems. In *Proceedings of the 10th European Conference on Artificial Intelligence*, ECAI '92, pages 31–35, New York, NY, USA, 1992. John Wiley & Sons, Inc.

Gent, I. P., Petrie, K. E., and Puget, J.-F. Symmetry in Constraint Programming. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, chapter 10, pages 329–376. Elsevier, 2006. doi: 10.1016/S1574-6526(06)80014-3.

Giemsa, G. Eine Vereinfachung und Vervollkommnung meiner Methylenblau-Eosin-Faerbemethode zur Arzielung der Romanowsky-Nochtschen Chromatinfaerbung. *Centralblatt fuer Bakteriologie I Abteillung*, 32:307–313, 1904.

Giles, K. and van Hoeve, W. Solving a Supply-Delivery Scheduling Problem with Constraint Programming. *Lecture Notes in Computer Science*, 9892, 2016. doi: 10.1007/978-3-642-29828-8.

Godard, D., Laborie, P., and Nuijten, W. Randomized Large Neighborhood Search for Cumulative Scheduling. *Proc. of ICAPS*, pages 81–89, 2005.

González, M. A., Oddi, A., Rasconi, R., and Varela, R. Scatter search with path relinking for the job shop with time lags and setup times. *Computers & Operations Research*, 60:37–54, 2015. doi:

10.1016/j.cor.2015.02.005.

Hall, L. *Computational Complexity*, pages 238–241. Springer US, Boston, MA, 2013. doi: 10.1007/978-1-4419-1153-7_141.

Hartmann, S. and Briskorn, D. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010. doi: 10.1016/j.ejor.2009.11.005.

Heinz, S. *Presolving techniques and linear relaxations for cumulative scheduling*. PhD thesis, Technical University of Berlin, 2018.

Herroelen, W. and Leus, R. Robust and reactive project scheduling: a review and classification of procedures. *International Journal of Production Research*, 42(8):1599–1620, 2004. doi: 10.1080/00207540310001638055.

Hnich, B., Smith, B. M., and Walsh, T. Dual Modelling of Permutation and Injection Problems. *J. Artif. Int. Res.*, 21(1):357–391, feb 2004.

Hochbaum, D. S. and Shamir, R. Strongly polynomial algorithms for the high multiplicity scheduling problem. *Operations Research*, 39(4):648–653, 1991.

Homberger, J. A multi-agent system for the decentralized resource-constrained multi-project scheduling problem. *International Transactions in Operational Research*, 14:565–589, 2007.

Hooke, R. *Micrographia: or Some Physiological Descriptions of Minute Bodies, Made by Magnifying Glasses with Observations and Inquiries Thereupon*. James Allestry, 1665.

Hurink, J. L., Kok, A. L., and Paulus, J. J. Decomposition Method for Project Scheduling with Spatial Resources. pages 1–15, 2002.

Jaffar, J. and Lassez, J.-L. Constraint Logic Programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '87, pages 111–119, New York, NY, USA, 1987. ACM. doi: 10.1145/41625.41635.

Kaplan, L. *Resource-constrained project scheduling with preemption of jobs*. Unpublished ph.d. dissertation, University of Michigan, USA, 1998.

Kimms, A. *Mathematical Programming and Financial Objectives for Scheduling Projects*. PhD thesis, University of Melbourne, 2010.

Kinable, J., Wauters, T., and Vanden Berghe, G. The concrete delivery problem. *Computers and Operations Research*, 48:53–68, 2014. doi: 10.1016/j.cor.2014.02.008.

Kinable, J. A Reservoir Balancing Constraint with Applications to Bike-Sharing. *Integration of AI and OR Techniques in Constraint Programming*, 9075(May):288–305, 2015. doi: 10.1007/978-3-319-18008-3.

Kinnunen, T. *Cost-efficient vacation planning with variable workforce demand and manpower*. PhD thesis, Aalto University, 2016.

Kizilay, D., Eliiyi, D. T., and Hentenryck, P. V. Constraint and Mathematical Programming Models for Integrated Port Container Terminal Operations. In *Proceedings of the 15th international conference on the integration ofconstraint programming, artificial intelligence, and operations research (CPAIOR 2018).*, 2017.

Kiziltan, Z. *Symmetry Breaking Ordering Constraints*. PhD thesis, Uppsala University, 2004.

Klein, R. Bidirectional planning: Improving priority rule-based heuristics for scheduling resource-

constrained projects. *European Journal of Operational Research*, 127(3):619–638, 2000. doi: 10.1016/S0377-2217(99)00347-1.

Kolisch, R. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2):320–333, 1996. doi: 10.1016/0377-2217(95)00357-6.

Kolisch, R. *Shifts, Types, and Generation Schemes for Project Schedules*, volume 1. Springer International Publishing, 2015. doi: 10.1007/978-3-319-05443-8.

Kolisch, R. and Hartmann, S. *Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis*, pages 147–178. Springer US, Boston, MA, 1999. doi: 10.1007/978-1-4615-5533-9_7.

Kolisch, R. and Hartmann, S. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23–37, 2006. doi: 10.1016/j.ejor.2005.01.065.

Kolisch, R. and Sprecher, A. PSPLIB - A project scheduling problem library. *European Journal of Operational Research*, 96(1):205–216, 1997. doi: 10.1016/S0377-2217(96)00170-1.

Koné, O., Artigues, C., Lopez, P., and Mongeau, M. Event-based MILP models for resource-constrained project scheduling problems. *Computers and Operations Research*, 38:3–13, 2011. doi: 10.1016/j.cor.2012.10.018.

Kopanos, G. M., Kyriakidis, T. S., and Georgiadis, M. C. New continuous-time and discrete-time mathematical formulation for resource-constrained project scheduling problems. *Computers and Chemical Engineering*, 68:96–106, 2014. doi: 10.1016/j.compchemeng.2014.05.009.

Kovács, A. and Váncza, J. Exploiting Repetitive Patterns in Practical Scheduling Problems. In *43rd CIRP International Conference on Manufacturing Systems*, pages 868–875, Vienna, Austria, 2010.

Kovács, A. and Váncza, J. Progressive Solutions : A Simple but Efficient Dominance Rule for Practical RCPSP. In Beck, J. C., , and Smith, B. M., editors, *In Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2006*, pages 139–151. Springer Berlin Heidelberg, 2006. doi: https://doi.org/10.1007/11757375_13.

Kreter, S., Schutt, A., and Stuckey, P. J. Using constraint programming for solving RCPSP/max-cal. *Constraints*, 22(3):432–462, 2017. doi: 10.1007/s10601-016-9266-6.

Laborie, P. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143(2):151–188, 2003a. doi: 10.1016/S0004-3702(02)00362-4.

Laborie, P. Resource temporal networks: Definition and complexity. *IJCAI International Joint Conference on Artificial Intelligence*, pages 948–953, 2003b.

Laborie, P. and Rogerie, J. Reasoning with Conditional Time-Intervals. *Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, May 15-17, 2008, Coconut Grove, Florida, USA*, pages 555–560, 2008.

Laborie, P. and Rogerie, J. Temporal linear relaxation in IBM ILOG CP Optimizer. *Journal of Scheduling*, 19(4):391–400, 2016. doi: 10.1007/s10951-014-0408-7.

Laborie, P., Rogerie, J., Shaw, P., and Vilim, P. Reasoning with Conditional Time-Intervals. Part II: An Algebraical Model for Resources. *FLAIRS Conference*, pages 201–206, 2009.

Laborie, P., Rogerie, J., Shaw, P., and Vilím, P. IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG. *Constraints*, 23(2):210–250, 2018. doi: 10.1007/

s10601-018-9281-x.

Lahrichi, A. Scheduling: the Notions of Hump, Compulsory Parts and their Use in Cumulative Problems. *C. R. Acad. Sci. Paris*, pages 209–211, 1982.

Law, Y. C. and Lee, J. H. M. Model Induction : a New Source of CSP Model Redundancy From Viewpoints to CSP Models. In *AAAI-02*, pages 54–60, 2002.

Lenstra, J. K., Rinnooy Kan, A. H., and Brucker, P. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1(C):343–362, 1977. doi: 10.1016/S0167-5060(08)70743-X.

Lhomme, O. Consistency techniques for numeric CSPs. *International Joint Conference on Artificial Intelligence*, 13:232–232, 1993.

Lombardi, M. *Hybrid Methods for Resource Allocation and Scheduling Problems in Deterministic and Stochastic Environments*. PhD thesis, Universita di Bologna, 2009.

Loong, S., Ku, W., and Beck, J. Q-bounds consistency for the spread constraint with variable mean. *Constraints*, 21(646), 2016.

Lopez-Ortiz, A., Quimper, C. G., Tromp, J., and Van Beek, P. A fast and simple algorithm for bounds consistency of the all different constraint. *IJCAI International Joint Conference on Artificial Intelligence*, pages 245–250, 2003.

Luby, M., Sinclair, A., and Zuckerman, D. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993. doi: 10.1016/0020-0190(93)90029-9.

Manier, M. A. and Bloch, C. A classification for hoist scheduling problems. *International Journal of Flexible Manufacturing Systems*, 15(1):37–55, 2003. doi: 10.1023/A:1023952906934.

Margot, F. Symmetry in Integer Linear Programming. In Juenger, M., Naddef, D., Pulleyblank, W. R., Rinaldi, G., Liebling, T. M., Nemhauser, G. L., Reinelt, G., and Wolsey, L. A., editors, *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, chapter 17, pages 647–686. Springer, Berlin, Heidelberg, 2010. doi: 10.1007/978-3-540-68279-0.

Marriott, K. and Stuckey, P. *Programming with constraints: an introduction*. MIT Press, 1998.

Masin, M. and Raviv, T. Linear programming-based algorithms for the minimum makespan high multiplicity jobshop problem. *Journal of Scheduling*, 17(4):321–338, 2014. doi: 10.1007/s10951-014-0376-y.

Mayer, P. Ueber das Farben mit Hamatoxylin. *Mitt Zool Stat Neapel*, 10:170–86, 1891.

Melchiors, P. *Dynamic and Stochastic Multi-Project Planning*. Lecture Notes in Economics and Mathematical Systems, 2015. doi: 10.1007/978-3-319-04540-5.

Mercier, L. and Van Hentenryck, P. Strong polynomiality of resource constraint propagation. *Discrete Optimization*, 4(3-4):288–314, 2007. doi: 10.1016/j.disopt.2007.01.001.

Michel, L. and Van Hentenryck, P. Activity-based search for black-box constraint programming solvers. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7298 LNCS:228–243, 2012. doi: 10.1007/978-3-642-29828-8_15.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., and Bianco, L. An Exact Algorithm for the Resource-Constrained Project Scheduling Problem Based on a New Mathematical Formulation. *Management Science*, 44(5):714–729, 1998. doi: 10.1287/mnsc.44.5.714.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC 2001)*, pages 530–535,

Las Vegas, NV, USA, 2001. IEEE. doi: http://doi.acm.org/10.1145/378239.379017.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. MiniZinc: Towards a Standard CP Modelling Language. *Principles and Practice of Constraint Programming – CP 2007*, pages 529–543, 2007. doi: 10.1007/978-3-540-74970-7_38.

Neumann, K. and Zhan, J. Heuristics for the minimum project-duration problem with minimal and maximal time lags under fixed resource constraints. *Journal of Intelligent Manufacturing*, 6(2): 145–154, 1995. doi: 10.1007/BF00123686.

Neumann, K. and Schwindt, C. Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, 56(3):513–533, 2003. doi: 10.1007/s001860200251.

Neumann, K. and Zimmermann, J. *Methods for Resource-Constrained Project Scheduling with Regular and Nonregular Objective Functions and Schedule-Dependent Time Windows*, pages 261–287. Springer US, Boston, MA, 1999. doi: 10.1007/978-1-4615-5533-9_12.

Nuijten, W. P. and Aarts, E. H. A computational study of constraint satisfaction for multiple capacitated job shop scheduling. *European Journal of Operational Research*, 90(2):269–284, 1996. doi: 10.1016/0377-2217(95)00354-1.

Ohrimenko, O., Stuckey, P. J., and Codish, M. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009. doi: 10.1007/s10601-008-9064-x.

Pape, C. L. Implementation of resource constraints in ILOG SCHEDULE: a library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering*, 3(2):55, 1994. doi: 10.1049/ise.1994.0009.

Pape, C. L., Couronné, P., Vergamini, D., and Gosselin, V. Time-versus-capacity compromises in project scheduling. *In Proc. of the 13th Workshop of the UK Planning Special Interest Group*, pages 1–13, 1994.

Petrovic, D., Castro, E., Petrovic, S., and Kapamara, T. *Radiotherapy Scheduling*. 2013. doi: 10.1007/978-3-642-39304-4.

Policella, N., Smith, S. F., and Oddi, A. Generating Robust Schedules through Temporal Flexibility. In *ICAPS-04 Proceedings*, pages 209–218, 2004.

Prichard, J. W. Overview of automated immunohistochemistry. *Archives of Pathology and Laboratory Medicine*, 138(12):1578–1582, 2014. doi: 10.5858/arpa.2014-0083-RA.

Pritsker, A. and Watters, L. A zero-one programming approach to scheduling with limited resources. *The RAND Corporation*, 1(RM-5561-PR), 1968.

Pritsker, A. A. B., Waiters, L. J., and Wolfe, P. M. Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach. *Management Science*, 16(1):93–108, 1969. doi: 10.1287/mnsc.16.1.93.

Puget, J.-F. Constraint Programming Next Challenge : Simplicity of Use. In *Principles and Practice of Constraint Programming*, pages 5–8, 2004.

Rahimi, A., Karimi, H., and Afshar-Nadjafi, B. Using meta-heuristics for project scheduling under mode identity constraints. *Applied Soft Computing Journal*, 13(4):2124–2135, 2013. doi: 10.1016/j.asoc.2012.11.002.

Rasmussen, M. S., Justesen, T., Dohn, A., and Larsen, J. The Home Care Crew Scheduling Problem: Preference-based visit clustering and temporal dependencies. *European Journal of Operational Research*, 219(3):598–610, 2012. doi: 10.1016/j.ejor.2011.10.048.

Refalo, P. Impact-Based Search Strategies for Constraint Programming. *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming*, 3258:557–571, 2004. doi: 10.1007/978-3-540-30201-8_41.

Regin, J. C. A filtering algorithm for constraints of difference. In *Proceedings of the 12th National Conference of the American Association for Artificial Intelligence*, pages 362–367, 1994.

Riise, A., Mannino, C., and Burke, E. K. Modelling and solving generalised operational surgery scheduling problems. *Computers and Operations Research*, 66:1–11, 2016. doi: 10.1016/j.cor.2015.07.003.

Rossi, F., Beek, P. V., and Walsh, T. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. 2006.

Sankaran, J. K., Bricker, D. L., and Juang, S. H. Strong fractional cutting-plane algorithm for resource-constrained project scheduling. *International Journal of Industrial Engineering : Theory Applications and Practice*, 6(2):99–111, 1999.

Schutt, A. and Stuckey, P. J. Explaining Producer/Consumer Constraints. In Rueher, M., editor, *Principles and Practice of Constraint Programming: 22nd International Conference*, Lecture Notes in Computer Science, pages 438—-454. Springer International Publishing, 2016. doi: 10.1007/978-3-319-44953-1_28.

Schutt, A., Feydy, T., Stuckey, P. J., and Wallace, M. G. Explaining the cumulative propagator. *Constraints*, 16(3):250–282, 2011. doi: 10.1007/s10601-010-9103-2.

Schutt, A., Feydy, T., and Stuckey, P. J. Scheduling optional tasks with explanation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8124 LNCS:628–644, 2013a. doi: 10.1007/978-3-642-40627-0_47.

Schutt, A., Feydy, T., and Stuckey, P. J. Explaining Time-Table-Edge-Finding Propagation for the Cumulative Resource Constraint. *Lecture Notes in Computer Science*, pages 234–250, 2013b.

Schutt, A., Feydy, T., Stuckey, P. J., and Wallace, M. G. Solving RCPSP/max by lazy clause generation. *Journal of Scheduling*, 16(3):273–289, 2013c. doi: https://doi.org/10.1007/s10951-012-0285-x.

Schwindt, C. and Zimmermann, J. *Handbook on Project Management and Scheduling*, volume 1. Springer International Publishing, 2015a. doi: 10.1007/978-3-319-05443-8.

Schwindt, C. and Zimmermann, J. *Handbook on Project Management and Scheduling*, volume 2. Springer International Publishing, 2015b. doi: 10.1007/978-3-319-05915-0.

Shaw, P. Using constraint programming and local search methods to solve vehicle routing problems. *Principles and Practice of Constraint Programming - Cp98*, 1520:417–431, 1998. doi: 10.1007/3-540-49481-2_30.

Shaw, P., Ilog, S. A., and Hb, L. T. A Constraint for Bin Packing. pages 648–649, 2004.

Smith, B. M. Modelling. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 377–406. Elsevier, 2006. doi: 10.1016/S1574-6526(06)80015-5.

Sprecher, A., Kolisch, R., and Drexl, A. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal Of Operational Research*, 80(1):94–102, 1995. doi: 10.1016/0377-2217(93)E0294-8.

Tesch, A. *Compact Models for the Resource-Constrained Project Scheduling Problem*. PhD thesis, Technische Universitat Berlin, 2016.

Tesch, A. Improved Compact Models for the Resource-Constrained Project Scheduling Problem. In Fink, A., F{\"u}genschuh, A., and Geiger, M. J., editors, *Operations Research Proceedings*, Operations Research Proceedings (GOR (Gesellschaft für Operations Research e.V.)), pages 25–30. Springer, Cham, 2018a. doi: 10.1007/978-3-319-55702-1_4.

Tesch, A. Improving Energetic Propagations for Cumulative Scheduling. *ZIB Report*, 29(June), 2018b.

Toffolo, T. A., Santos, H. G., Carvalho, M. A., and Soares, J. A. An integer programming approach to the multimode resource-constrained multiproject scheduling problem. *Journal of Scheduling*, 19(3):295–307, 2016. doi: 10.1007/s10951-015-0422-4.

Van Der Veen, J. A. A. and Zhang, S. Low-complexity algorithms for sequencing jobs with a fixed number of job-classes. *Computers & Operations Research*, 23(11):1059–1067, 1996.

Verma, S. and Dessouky, M. Multistage Hybrid Flowshop Scheduling with Identical Jobs and Uniform Parallel Machines. *Journal of Scheduling*, 2:135–150, 1999. doi: 10.1016/S0377-2217(97)00194-X.

Vilím, P. Edge Finding Filtering Algorithm for Discrete Cumulative Resources in O(knlog(n)). In *Principles and Practice of Constraint Programming-CP ...*, Lecture Notes in Computer Science, vol 6697, pages 802–816. Springer, Berlin, Heidelberg, 2009. doi: https://doi.org/10.1007/978-3-642-21311-3_22.

Vilím, P. *Global Constraints in Scheduling*. PhD thesis, Charles University, 2007.

Vilím, P. Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources. In Achterberg, T. and Beck, J. C., editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 230–245, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

Vilim, P., Laborie, P., and Shaw, P. Failure-Directed Search for Constraint-Based Scheduling. *Integration of AI and OR Techniques in Constraint Programming: 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*, pages 437–453, 2015. doi: 10.1007/978-3-642-38171-3.

Voß, S. and Witt, A. Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: A real-world application. *International Journal of Production Economics*, 105(2):445–458, 2007. doi: 10.1016/j.ijpe.2004.05.029.

Wallace, M. Practical Applications of Constraint Programming. *Constraints*, 1(1-2):139–168, 1996. doi: 10.1007/BF00143881.

Watt, A. and Eng, N. Chapter 11 Functional Dependencies. In *Database Design*, pages 1–10. Victoria, B.C.: BCcampus., 2nd editio edition, 2014.

Wikum, E. D., Llewellyn, D. C., and Nemhauser, G. L. One-machine generalized precedence constrained scheduling problems. *Operations Research Letters*, 16(2):87–99, 1994. doi: 10.1016/0167-6377(94)90064-7.

Williams, R., Gomes, C. P., and Selman, B. Backdoors To Typical Case Complexity. In *18th Intl. Joint Conf. on Aritificial Intelligence (IJCAI-03)*, 2003.

Yu, W., Hoogeveen, H., and Lenstra, J. K. Minimizing makespan in a two machine flow shop with delays and unit time operations is NP-hard. *Journal of Scheduling*, 7(5):333–348, 2004.

Zapata, J. C., Hodge, B. M., and Reklaitis, G. V. The multimode resource constrained multiproject scheduling problem: Alternative formulations. *AIChE Journal*, 54(8):2101–2119, 2018. doi: 10.

1002/aic.11522.

Zhan, J. Heuristics for scheduling resource-constrained projects in MPM networks. *European Journal of Operational Research*, 76(1):192–205, 1994. doi: 10.1016/0377-2217(94)90016-7.

Zhang, H. and Stickel, M. E. An efficient algorithm for unit propagation. In *In Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics (AI-MATH'96), Fort Lauderdale (Florida USA*, pages 166—-169, 1996.

Zhao, C., Fu, J., and Qu, Q. Real-Time Dynamic Hoist Scheduling for Multistage Material Handling Process Under Uncertainties. *American Institute of Chemical Engineers*, 59(2):465–482, 2013. doi: 10.1002/aic.