Optimising the design of buffer preparation in bioprocessing facilities

Sean Tully, M.A. (*Cantab.*), M.Eng.

August 29, 2017

A dissertation submitted to University College Dublin in part fulfilment of the requirements of the degree of M.Sc. in Business Analytics

Michael Smurfit Graduate School of Business,

University College Dublin

August, 2017

Supervisor: Prof. M. O'Neill

Head of School: Professor Ciarán Ó hÓgartaigh

Dedication

To Seona.

Contents

List of figures **vi**

List of tables **vii**

1 Introduction **1**

1.1 Bioprocessing 1

1.2 Bioprocess Engineering 2

1.3 Upstream and Downstream 3

1.4 Buffers and Media 3

1.5 Buffers and Media Preparation 4

1.6 Design of Buffer Preparation Areas 7

1.7 Problem Definition 8

2 Literature Review **11**

2.1 Introduction 11

2.2 Bioprocess Facility Design 11

2.3 Facility Design Optimisation in the Process Industry 13

2.4 Simulation and Optimisation 14

3 Data **17**

3.1 Introduction 17

3.2 Vessel Data 18

3.3 Buffer Data 20

3.4 Parameters 22

3.5 Data Sources 25

3.5.1 Random Data 25

3.5.2 Real-World Data 26

4 Methodology 27

4.1 Introduction 27

4.2 Slots 28

4.3 Objective Function 29

4.4 Basic Model 29

4.4.1 Buffers Dedicated to Slots 30

4.4.2 Vessel Instances Dedicated to Slots 30

4.4.3 Vessel Capacity 30

4.4.4 Preparation Vessel Utilisation 32

4.4.5 Basic Model Summary 33

4.5 Complete Model 34

4.5.1 Hold Procedure Duration 34

4.5.2 Introducing the Scheduling Constraint 35

4.5.3 Pairs of Distinct Buffers Prepared in the Same Slot . . . 36

4.5.4 Absolute value constraints 37

4.5.5 Preparation Procedure Reference Times 39

4.5.6 Dealing with Cyclic Time Windows 40

4.5.7 Preparation Scheduling 44

4.5.8 Complete Model Summary 45

4.6 Further Optimisation 48

4.6.1 Goal programming 48

4.6.2 Secondary Model Summary 50

4.7 Implementation 52

4.7.1 MILP Solvers 53

4.7.2 Development Environment 54

5 Results 56

5.1 Output Data 56

5.2 Solution Duration 59

5.2.1 Variation of Problem Size 59

5.2.2 Comparison of Solvers 61

5.2.3 Real-World Data 63

6 Discussion **64**

6.1 Comment on Results 64

6.2 Challenges 65

6.3 Complexity 65

7 Conclusions **68**

Appendices

Program installation and usage **71**

Program code description **73**

Real-world data set 1 **78**

Real-world data set 2 **81**

Bibliography **85**

List of Notation **88**

List of Figures

1.1 Single buffer preparation process flow diagram 5

3.1 Equipment time utilisation for a single buffer 20

4.1 Example feasible region for buffer k scheduling 40
4.2 All feasible region cases for buffer k scheduling 41
4.3 Random example – primary objective (note that buffers are individually coloured; each buffer has a dedicated hold vessel, from which its colour may be determined) 49
4.4 Random example – secondary objective 51

5.1 Equipment time utilisation for random example 58
5.2 Solution time as a function of buffer count using CPLEX 60
5.3 Solution time as a function of buffer count and solver 62

6.1 Model complexity – variables 67
6.2 Model complexity – equations 67

List of Tables

3.1 Vessel data for random example 18
3.2 Buffer data for random example 20
3.3 Global parameters for random example 23

4.1 Truth table for $\boldsymbol{w}_{nkp}$ 36
4.2 Truth table for $\boldsymbol{a}_{nk}$ 37
4.3 Truth table for $\boldsymbol{\beta} - \boldsymbol{\alpha}$ 38
4.4 Truth table for $\boldsymbol{q}_n$ 39
4.5 Truth table for $\boldsymbol{r}_n$ 43
4.6 Truth table for $\boldsymbol{s}_n$ 43
4.7 Truth table for $\boldsymbol{u}_n$ 44
4.8 Truth table for equation 4.5.29 45
4.9 Installed third-party python libraries 55

5.1 Required preparation vessels for random example 56
5.2 Buffer / vessel matrix for random example 57

6.1 Model complexity 65
6.2 Simplified model complexity 65

Preface

The cold smell of potato mould, the squelch and slap

Of soggy peat, the curt cuts of an edge

Through living roots awaken in my head.

But I've no spade to follow men like them.

Between my finger and my thumb

The squat pen rests.

I'll dig with it.

— Seamus Heaney, *Digging*

The motivation for this thesis was a desire to have a robust and efficient method for optimising a problem encountered when attempting early-stage design of bioprocessing facilities.

Chapter 1 begins with an overview of the field of bioprocessing, followed by an outline of the design challenges that exist and an outline of the problem we are trying to solve; the buffer vessel selection problem.

In Chapter 2, the relevant literature is reviewed, starting with topics pertaining to bioprocess facility design and branching into the broader topics of process optimisation and linear programming.

Chapter 3 outlines the form of the input data and describes the input parameters to the mathematical model.

In Chapter 4, several mixed-integer models are developed and the code used to implement them is discussed.

Chapter 5 catalogues the results obtained. This includes outlining the form of the results, visualising the results and investigation into the time taken to obtain a solution.

Chapter 6 discusses the results, and the complexity of the problem.

Chapter 7 outlines the conclusions drawn from the exercise and the scope for further work.

University College Dublin Sean Tully
August 29, 2017

Acknowledgements

I would like to thank David Lynch for reviewing many iterations of this document and providing much helpful feedback.

I would also like to thank Dr. Paula Carroll for her advice on linear programming during the early stages of development.

I would also like to acknowledge my supervisor, Prof. Michael O'Neill.

Abstract

This dissertation seeks to define the problem of buffer vessel selection in bioprocessing facility design. It succeeded in modelling the problem mathematically, as a mixed integer linear programming problem. The problem was then implemented such that it can be solved using one of a range of mixed-integer linear programming packages. Equipment time utilisation plots were designed to visualise the results. The models were trialled on both random and real-world data and found to be robust. The complexity of the models and the time taken for solution were investigated. Scope for further work was outlined in terms of some additional features or constraints that could be added to the model.

Keywords: ***Process engineering***, ***process design***, ***biotechnology***, ***linear programming***, ***mixed integer linear programming.***

Chapter 1

Introduction

Well I don't think we're *for* anything. We're just products of evolution. You can say, 'Gee, your life must be pretty bleak if you don't think there's a purpose.' But I'm anticipating having a good lunch.

— James Watson, *in conversation with Richard Dawkins*

1.1 Bioprocessing

Before the advent of biotechnology, most therapeutics (medicines) were what are now termed *small molecule* drugs. The pharmaceutical industry was concerned with the synthesis of these products via predominantly chemical processes, such as reaction, distillation and crystallisation. Small molecule drugs typically consist of tens or hundreds of atoms, such as paracetamol, which has a molar mass of approximately 151 g/mol, or aspirin (acetylsalicylic acid), which has a molar mass of approximately 180 g/mol.

With the advent of recombinant D.N.A technology, biochemists gained the ability to re-program the D.N.A. of simple biological microorganisms such as *Escherichia coli* and, eventually, mammalian cells, such as those of the Chinese Hamster (*Cricetulus griseus*). The genetic structure of these cells could be modified to produce complex molecules, which had previously proved difficult or impossible to synthesise by any other means. The biopharmaceutical industry is concerned with the synthesis of therapeutics via such biological pathways. These products are known by various names, such as *biopharmaceuticals*, *protein therapeutics* or, colloquially, as *biotech drugs*.

The first protein therapeutic to be synthesised on a large scale using biological pathways was insulin, which is a hormone used to regulate metabolism and is administered to individuals suffering from diabetes. Human insulin has a molar mass of approximately 5808 g/mol. It was not possible to commercially synthesise insulin chemically and it was initially produced by extracting the hormone from the pancreases of mammals such as cows or pigs. In 1978, scientists working at the American company Genentech (now a subsidiary of the Swiss pharmaceutical company F. Hoffmann-La Roche AG) successfully modified cells of *E. coli* to produce insulin and this synthetic insulin was first brought to market in 1982.

The industry that has grown up around the production of biopharmaceuticals is known as the *biopharmaceutical industry*, or, colloquially, as the *biotech* or *biopharma* industry. A report by strategy consultants McKinsey & Company (Otto *et al.*, 2014) estimates that the biopharmaceutical market had global revenues of $163 billion per annum and was worth 20% of the overall pharmaceutical market. Otto *et al.* (2014) also note that large-scale biopharmaceutical manufacturing facilities typically cost in the region of “$200 million to $500 million or more” to build.

1.2 Bioprocess Engineering

Schaschke (2014) defines bioprocess engineering as “A specialist branch of (chemical) engineering that involves the design and operation of processes used for the production of biological products such as foods, pharmaceuticals, and biopolymers.” The design of a large-scale biopharmaceutical facility typically takes about two years and requires a multidisciplinary team of engineers and scientists.

1.3 Upstream and Downstream

A complete facility typically starts with frozen vials of cells (the *working cell bank*) and finishes with the final formulated product in either bulk form or filled into its final packaging e.g. syringes. Facilities are nominally divided into *upstream* and *downstream* sections. The upstream section is predominantly concerned with the expansion of cells from a small vial into progressively larger tanks of *media*. The final stage of this growth occurs in the *production bioreactor*, wherein the conditions can be altered to encourage the cells to produce the target protein.

At the interface between upstream and downstream lies the *harvest* section. In the harvest section, some initial separation is performed to begin to isolate the target protein from the contents of the batch (which at this point include cells, cell waste, growth media, antibiotics and myriad other contaminants). At the end of the harvest section, all traces of the host cells should be removed and the batch is said to be *cell-free*. Different interpretations exist in the industry as to where the upstream-downstream split occurs, but it usually is defined as being at some point immediately before or after harvest.

Downstream processing is concerned with taking the cell-free but otherwise contaminated batch and purifying it through a series of orthogonal processes. Such processes can include filtration, ultrafiltration/diafiltration (*UF/DF*), chromatography, reaction, virus inactivation and formulation. At the end of downstream processing, a batch should consist of formulated bulk product, ready to be filled into its final packaging for delivery. The final fill/finish steps often occur in a separate, sterile facility.

1.4 Buffers and Media

Both upstream and downstream sections of a facility require the preparation of large volumes of aqueous solutions. Solutions used upstream are typically referred to as *media* and those used downstream are typically referred to as *buffers*. Strictly speaking, media refers to the solutions of nutrients into which cells are expanded, but the term is usually used to encompass all other upstream solutions, such as acids, bases and anti-foam used in the bioreactors. Strictly speaking, a chemist would define a buffer as a solution which maintains its pH over a wide range of concentrations. Most solutions used downstream do indeed meet this criteria, but the term *buffers* is generally used as a catch-all for all solutions used in the downstream section of a facility. A typical process to produce a *monoclonal antibody* (a common family of protein therapeutics) can use tens or hundreds of litres of buffers and media per litre volume in the production bioreactor. Typical large-scale production bioreactor volumes for such processes are in the range 10 000 l to 30 000 l. Each batch may use in the region of 20–40 different buffers and media.

1.5 Buffers and Media Preparation

One of the reasons for the catch-all definitions of *buffers* and *media* in the section above is to do with segregation. The upstream and downstream sections of the plant are segregated to prevent cross-contamination. As a result, there are typically two main areas where solutions are prepared. Buffers are prepared in an area called *buffer preparation*, for use downstream and media are prepared in an area called *media preparation*, for use upstream. There may also be a separate area for preparing sterile buffers for the final formulation, again to reduce the possibility of contamination and ensure sterility is maintained. In both media and buffer preparation, a preparation vessel is used to prepare the solution and then it is typically sterile filtered into a separate hold vessel.

Figure 1.1 is a *process flow diagram*; a simplified schematic that shows the main equipment used in buffer preparation.

Note that we define a complete processing step in a piece of equipment as a *procedure*, e.g. ‘preparation of buffer n’. Each procedure consists of one or more *operations*, e.g. ‘charge solids to vessel’. These definitions will be used throughout this document.

A clean buffer preparation vessel is first charged with *WFI* (water for injection) – an extremely pure quality water. Solids are also charged to the vessel. After

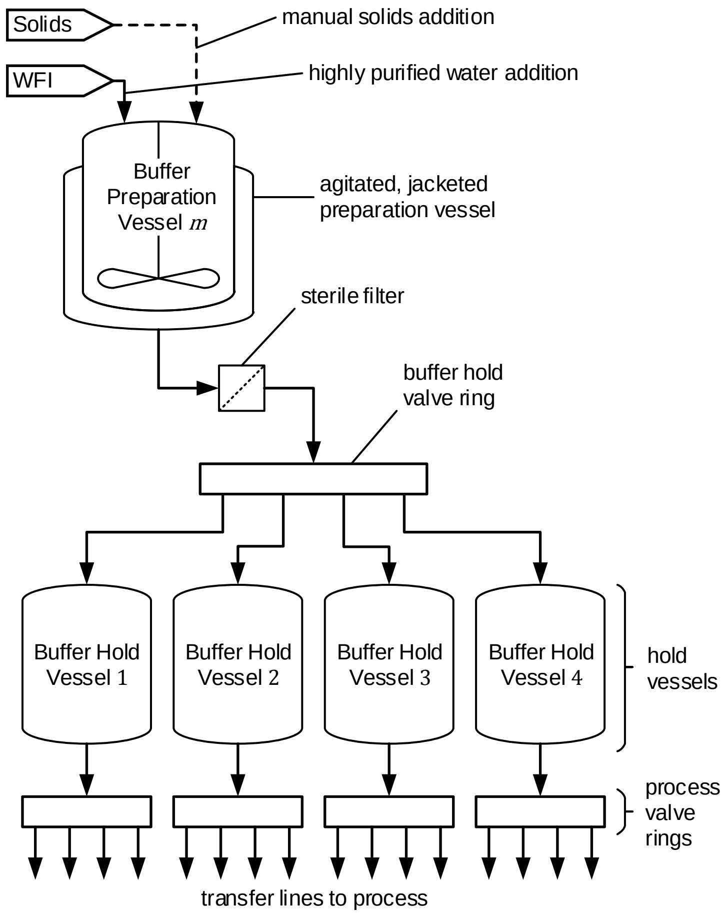


Figure 1.1: Single buffer preparation process flow diagram

closing the solids charging port, the contents of the vessel are agitated to ensure complete solution of the solids. Heating or cooling may be applied via the vessel jacket. Sampling or testing may be performed, and adjustments made by adding more solids or WFI.

Once the buffer has passed all required quality checks, it is pressure-transferred, via a sterile filter, to a sterile hold vessel. Prior to this point, the sterile filter, the hold vessel and all interconnecting pipework will have been sterilised, using clean steam, and then dried. It is not possible to maintain sterility in the preparation vessel, since open solids additions are required, so the sterile filter acts as a barrier between the clean (but not sterile) preparation vessel and the sterilised hold vessel. Transfer of the buffer is usually performed by pressurising the preparation vessel using sterile, filtered nitrogen (pumps are not desirable since they contain complex interior surfaces that are hard to sterilise). If the preparation vessel is capable of serving more than one hold vessel, the buffer is directed via a *valve ring* to its destination vessel. A valve ring is simply a loop of pipe. The valve ring has an open inlet port and several outlet ports. Each outlet port has a specialised valve, called a *zero dead-leg* valve, which is located right at its junction with the loop. The effect of this setup is that there are no points in the loop where liquid can stagnate, which again aids in maintaining sterility.

Once the preparation vessel has finished its transfer of the buffer, it is cleaned, ready for re-use. The vessel is cleaned, along with some of its pipework, in an automated manner known as *CIP* (clean-in-place). A piece of equipment known as a *CIP skid* is used to perform the CIP. A CIP Skid is connected to purified water and/or WFI supplies, as well as to sources of cleaning chemicals, such as acids, bases and detergents. The CIP skid contains a system of vessels, temperature control units and pumps which are capable of sending these various solutions to destination equipment in predefined cycles. The waste generated during CIP is also returned to the CIP skid via scavenging pumps, which direct waste to other waste handling systems for further treatment or storage.

The sterilised hold vessel, having received the buffer, holds it until it is required by the process. If the buffer is made ahead of time, it may be held for several hours before first use. A buffer may be used by one or more operations in one or more procedures in the production process. Accordingly, it may be drawn discontinuously from the hold vessel during the course of a batch.

Once the last use of the buffer has finished, the buffer hold vessel undergoes a CIP. The CIP leaves the vessel in a clean, but not sterile, state. Just before the vessel is to be used again, it undergoes *SIP* (steam-in-place); clean steam is passed through the vessel and pipework until it reaches a predefined temperature for a predefined hold time, after which the vessel is allowed to cool down to room temperature. Once the SIP is complete, the hold vessel is ready to receive the next lot of buffer.

1.6 Design of Buffer Preparation Areas

If a design for a large-scale production facility is to be carried out, the client must provide some information on the nature of the production process. Sometimes this can come in the form of information from another production facility, where the product is already being produced. It may also come directly from small-scale development work carried out in a laboratory, or a *pilot plant* (a small plant used to demonstrate how a process may be scaled from the laboratory to production scale). Given this information, a process engineer can create a *mass balance*, which is a document containing calculated values for all masses (and volumes) of products, additions and wastes for each procedure in the process. The mass balance is used to size the main production equipment (bioreactors, purification equipment, etc.). This mass balance provides, amongst other things, the volumes of all buffers and media required to make a batch. Two optimisation problems now emerge; how do we design the buffer and media preparation areas? For media, the problem is relatively easy to solve with some trial and error, since there are typically only about ten media used per batch and they often differ vastly in scale; the initial bioreactor may be 20 l in volume and the production bioreactor may be 20 000 l in volume. Because of this, the sizing of preparation vessels usually proceeds by picking a vessel capable of preparing the largest medium, defining a minimum fill volume and seeing what else can be prepared in it, then defining another vessel, and so on until sufficient vessels are defined. Media hold vessels, if required, may be similarly defined. The design of media preparation is usually relatively insensitive to schedule.

The problem of designing a buffer preparation area is more difficult to solve. There may be 20 or more different buffer compositions. Often each buffer is used multiple times in the same procedure or multiple times across multiple procedures. In procedures such as chromatography, somewhere in the region of 5–10 buffers may be needed in rapid succession. These chromatography buffers tend to be of similar volumes, so an efficient solution will look to maximise the number of buffers prepared in a given preparation vessel. If several similarly-sized buffers are to be prepared in the same preparation vessel, but all are required by the process concurrently, each will require a dedicated hold vessel and several of them will have to be prepared ahead of time, which puts pressure on the scheduling of their hold vessel activities. There is thus a trade-off between the numbers and volumes of vessels and the flexibility in scheduling buffers.

In practice, in the design of a buffer preparation area, there exists trade-offs between efficiency and flexibility, capital and operating costs, and many other factors such as installed area, installed volume, operability, layout/adjacency, pipework complexity and cleanability.

Other factors that affect the design of the area are choices between fixed equipment (typically in stainless steel) or disposable equipment (disposable plastic preparation and hold bags are currently available in volumes up to 5000 l), and material compatibility issues (high-nickel alloys, several times the cost of stainless steel, may be required for some buffers to prevent corrosion).

1.7 Problem Definition

The task of designing a buffer preparation area is complex. Current workflows are largely based on trial-and-error methods using a process engineering scheduling software package (a selection of which are detailed in Section 2.2). Using a chosen software package, a model is developed which captures the scheduling of the main process procedures and also the scheduling of the preparation and hold procedures for both buffer and media. Typically, a conservatively large array of buffer vessels is chosen and the schedule scenario is run. If there is an individual vessel for each task, the scenario will resolve easily, but the capital and space requirements of such a design will be sub-optimal. Via trial-and-error, individual preparation vessels may be removed or resized and the model re-run. The above process is iterated until it becomes difficult or impossible to remove or reduce the preparation vessels any further and obtain a feasible solution without scheduling clashes. At this point, iteration stops. It may appear that a 'good' solution has been found at this point, but there is no clear methodology to ensure that the solution is optimal. In the early feasibility or concept stages of a project, this process is cumbersome, the end points are poorly defined and any development of the underlying process which varies the volumes required may necessitate starting the optimisation again from scratch. An additional factor is that a working solution may exist for a given configuration, but the scheduling software is unable to resolve the problem, giving a false negative. The scheduling tools used for the process tend to be deterministic, rather than stochastic (although some sensitivity analysis is usually built in as an add-on); this makes it difficult to have confidence that a working schedule can handle the real-world batch-to-batch variability inherent in a process that depends on the productivity of living cells.

A more streamlined methodology for solving this optimisation problem is required and this dissertation is concerned with developing such a methodology and a software tool to implement it. The aims are as follows:

- Start with a reduced or basic case, including a number of simplifying assumptions
- Develop a tool to schedule the operations in buffer preparation and to vary the size and number of vessels and the preparation strategy to optimise the process with respect to some metric.

- Once a working framework has been developed, apply additional constraints (or remove simplifications) to provide a better approximation of real-world conditions.

In this study a technique is developed for design of the buffer preparation process, including vessel selection, sizing, assignment and scheduling. Unlike existing methods which utilise trial-and-error optimisation, the proposed tool provably produces an optimum solution. In addition, the algorithm converges to the optimum rapidly, for problems of typical real-world complexity, providing a much faster design. The proposed tool requires minimal information. The information is requested in a format that is easy to understand, making it accessible to process engineers without specific experience in using the complex scheduling tools outlined in Section 2.2.

This study provides a novel approach for solving a vessel assignment problem, and so contributes to the canon of knowledge in the areas of process engineering and linear programming.

The business imperatives are twofold. Firstly, for an engineering consultancy, the ability to rapidly, accurately and repeatably solve such problems gives a competitive edge, which can be used to win more business and to deliver more cost-effective design. Secondly. for a biopharmaceutical client, optimising this problem results in cost savings and having a well defined methodology for doing so gives confidence that an in-progress design is indeed optimal and robust.

Chapter 2

Literature Review

It was long before I got at the maxim, that in reading an old mathematician you will not read his riddle unless you plough with his heifer; you must see with his light, if you want to know how much he saw.

— Augustus de Morgan, *letter to W. R. Hamilton*

2.1 Introduction

In this chapter, recent papers related to the design of bioprocess facilities are reviewed. This is followed by the review of papers related to solving similar problems in related industries. Finally, papers relating to more general methodologies for solving scheduling and optimisation problems are reviewed.

2.2 Bioprocess Facility Design

Current bioprocess facility design leverages software packages for creating mass balances and for schedule simulation. The mass balance tool *SuperPro Designer*®[1] and the scheduling tool *SchedulePro*®[2], both from Intelligen, Inc.

[1] *SuperPro Designer*®, `http://www.intelligen.com/superpro_overview.html`

[2] *SchedulePro*®, `http://www.intelligen.com/schedulepro_overview.html`

(*Scotch Plains, New Jersey, U.S.A.*) are examples of software packages that are widely used, particularly by American firms. The *INOSIM*[3] family of software from INOSIM Software GmbH *Dortmund, Germany* is used particularly by German and Swiss firms.

Petrides *et al.* (2014) outline a workflow for the design of a "typical" monoclonal antibody facility using *SuperPro Designer*® and *SchedulePro*® and compare the use of these packages with other methodologies. While this paper does touch on buffer preparation, it does not elaborate on a strategy for optimising the area, stating:

> In most real processes, buffer scheduling is considerably more complex and challenging because of the larger number of buffers required for a typical process (more than twenty), the shared use of pipe segments and transfer panels, as well as constraints imposed by the limited availability of labor.

Petrides *et al.* (2014) cite an earlier paper by Toumi *et al.* (2010) which also looks at design of a facility for monoclonal antibody production. Toumi *et al.* (2010) also mention the difficulty of optimising the sizing and selection of buffer equipment, but do not outline a methodology for doing so.

Dietz *et al.* (2008) outlines a genetic algorithm approach to optimising the design of protein production facilities, which is capable of dealing with imprecise demands. It is primarily looking at the specification of the main process equipment and mass balance and it does not touch on buffer preparation.

No mentions could be found of methodologies for optimising buffer preparation in bioprocess facility design.

[3] *INOSIM*, `http://www.inosim-software.com/`

2.3 Facility Design Optimisation in the Process Industry

Casting the net wider to the pharmaceutical, chemical and other process industries yields several articles that deal with similar families of problems.

Of particular relevance to buffer preparation are efforts to simulate tank farm design (Al-Otaibi *et al.*, 2004; Stewart and Trierwiler, 2005; Sharda and Vazquez, 2009; Terrazas-Moreno *et al.*, 2012). Tank farms commonly exist in large pharmaceutical, chemical and oil & gas facilities and consist of arrays of tens of tanks that are usually dedicated to particular chemicals or products, but may be multi-use. These tanks are used to support the main process being carried out in the facility, similar to the role of buffer hold vessels. There are a great deal of papers that deal with optimising throughput or productivity of existing tank farms or existing batch processes, but very little articles relating to the optimisation of the design of the facilities required to run such processes.

Al-Otaibi *et al.* (2004) describe efforts to optimise the design of a tank farm for the oil & gas industry using both linear programming and Monte Carlo simulation. Their brief magazine article does not provide any detail on how the simulations were carried out.

Stewart and Trierwiler (2005) cite the work of Al-Otaibi *et al.* (2004) and outline a method for optimising tank farm design using Monte Carlo simulation. They mention the use of a software tool called *GRTMPS*[4] from *Haverley Systems, Inc., Houston, Texas, U.S.A*, supported by *Excel*[5] spreadsheet software and *Access*[6] database software (*Microsoft Inc, Redmond, Washington, U.S.A*). Again, theirs is a short magazine article. It indicates that the scheduling produced useful results but does not give any technical detail as to how the simulation was carried out.

Sharda and Vazquez (2009) outline the use of the discrete-event simulation

[4] *GRTMPS*, `https://www.haverly.com/main-products/13-products/9-grtmps`

[5] *Excel*, `https://office.microsoft.com/excel`

[6] *Access*, `https://office.microsoft.com/access`

tool *Arena*®[7] from *Rockwell Automation, Inc., Milwaukee, Wisconsin, U.S.A.* to optimise the utilisation of existing tank farm facilities and cite the work of Sharda and Vazquez (2009)

Terrazas-Moreno *et al.* (2012), citing Stewart and Trierwiler (2005) and Sharda and Vazquez (2009) provide a far more detailed description of efforts to optimise tank farm operations using mixed integer linear programming (*MILP*). Their work looks at optimising both schedule and tank selection. It is not strictly designed with the design of the facility itself, but rather the optimal operation of a designed facility.

The work of Terrazas-Moreno *et al.* (2012) provides some useful information on techniques that could be applied to solve the problem of buffer preparation area design, namely MILP and process scheduling. Branching out further into these fields yields more relevant literature.

Dedieu *et al.* (2003) suggests a hybrid approach using genetic algorithms and discrete-event simulation to address the problem of multi-objective batch plant design.

In Cavin *et al.* (2004, 2005), tabu search is discussed as a methodology to optimise the design of multi-purpose batch plants.

2.4 Simulation and Optimisation

Casting the net wider still to look at techniques for solving generalised scheduling and optimisation problems yields far more material, but further research is required to decide which techniques, if any, are relevant to the problem at hand.

There are two aspects to optimising the design. The first aspect is scheduling, as the ability of a tank to perform more than one task will depend on the times at which the tasks must (or may) occur. The second aspect is selecting the optimal sizes and numbers of vessels, which can be seen as a combinatorial optimisation problem.

[7] *Arena*, `https://www.arenasimulation.com/`

In his seminal 1957 paper, George Dantzig outlines several types of combinatorial optimisation problems, one of which is the *knapsack problem*. This problem relates to finding the most valuable selection of objects that can be carried in a knapsack, subject to a total weight limit, given a selection of candidate items, each having a weight and a value. A whole range of knapsack-type problems have been researched in the intervening period. Detailed descriptions of the most common problems and the research carried out over the half century following Dantzig's paper are given by Korte and Vygen (2012) and Martello and Toth (1990).

One knapsack-type problem that may be of particular relevance is the bin-packing problem. Martello and Toth (1990) describes the bin packing problem as one in which there are a number of items with associated weights and a number of bins with associated capacities. The aim is to assign each item to a bin so that the weight capacity of the bin is not exceeded and the number of bins is minimised. The concepts of *slots* introduced in Section 4.2 is somewhat analogous to the concept of bins in the bin-packing problem. A number of algorithms for both exact and approximate solutions of the bin-packing problem have been developed. Korte and Vygen (2012) state that the bin-packing problem is strongly **NP**-hard, indicating that efforts should be taken to minimise the sample space when dealing with optimising vessel selection.

Bettinelli *et al.* (2010) investigates a particular variant of the bin packing problem where there is a minimum filling constraint. This is an important consideration in vessel selection, usually some minimum fill level, e.g. 20–30% is defined so that the impeller in the vessel remains submerged during the mixing process and to reduce the volume of cleaning solutions or water required to clean the vessel as a fraction of the volume of buffer produced. The application of fill level constraints is discussed in Section 4.4.3. In terms of process scheduling methodologies, a number of papers exist in the field of chemical engineering (Ahmed and Sahinidis, 2000),

Ahmed and Sahinidis (2000) states that the general process planning problem is also **NP**-hard.

A detailed synopsis of scheduling methodologies applicable to the chemical and process industries is given by Harjunkoski *et al.* (2014).

Chapter 3

Data

> We have some freedom in setting up our personal standards of beauty, but it is especially nice when the things we regard as beautiful are also regarded by other people as useful.
>
> — Donald Knuth, *Computer Programming as an Art*

3.1 Introduction

This chapter will outline the requirements for input data.

For modelling a single process, a typical data-set will consist of three distinct files. The first file is a table of data relating to the available selection of vessels. The second file is a table of data relating to parameters specific to each buffer. The third file comprises a collection of global parameters that apply to all vessels and/or buffers.

One particular example, based on some randomly generated data, is used throughout this chapter to illustrate the relevant data formats.

Table 3.1: Vessel data for random example

names	volumes	costs
	V_m (l)	c_m (–)
1000 l	1000.0	63.10
2000 l	2000.0	95.64
3000 l	3000.0	121.98
4000 l	4000.0	144.96
5000 l	5000.0	165.72
6000 l	6000.0	184.88
8000 l	8000.0	219.71
10 000 l	10 000.0	251.19
12 000 l	12 000.0	280.83
16 000 l	16 000.0	333.02
18 000 l	18 000.0	357.41
20 000 l	20 000.0	380.73
22 000 l	22 000.0	403.14
25 000 l	25 000.0	435.28
30 000 l	30 000.0	485.59

3.2 Vessel Data

The vessel data will contain several parameters for each available vessel size, $m \in \mathcal{M}$. A vessel data-set will contain entries for M vessel sizes:

$$m \in \mathcal{M}; \quad \mathcal{M} = \{0, 1, 2, \ldots, m, \ldots, (M-1)\} \tag{3.2.1}$$

$$M = |\mathcal{M}| \tag{3.2.2}$$

Note that the computer scientists' convention, whereby counting starts at zero, is used throughout this report.

Typically, when designing a large-scale production facility, buffer preparation vessel volumes range from 1000 l to 30 000 l.

When ordering such a vessel, the stated size of the vessel is usually a nominal volume, which may differ from the liquid fill volume and may also differ from the maximum working volume. It is usual to round the stated or nominal volume to the nearest thousand litres.

For the purposes of this study, it is assumed that, for each vessel $m \in \mathcal{M}$, the specified vessel volume, V_m, is the maximum working volume (in litres) of the vessel. This is taken to be equivalent to the maximum volume of buffer which can be prepared in that vessel.

Vessels will also have a minimum working volume. It is usual to assume a minimum fill ratio of about 30% of the vessel volume. This limitation arises due to the minimum agitation volume of the impeller in the vessel. For the purposes of this simulation, the minimum fill ratio is a global parameter. It may be possible to specify a value of minimum fill ratio for every vessel size, but this level of detail is typically neither required nor available.

For each vessel, $m \in \mathcal{M}$, a (relative) cost, c_m, must also be defined. Note that absolute costs are not required to find the vessel selection that minimises costs. Vessel costs tend not to scale linearly with volume.

Sample vessel data are given in Table 3.1. In this table, the cost value for each vessel size has been estimated by raising the vessel volume (in litres) to a power of 0.6 and rounding to two decimal places. Note that the vessel volumes, V_m, are equal to the nominal volumes indicated in the 'names' column. The contents of the 'names' column are treated as identifiers and are not used in any calculation. It is not uncommon to have values of V_m slightly larger than their respective named (nominal) volumes; a vessel with a nominal volume of e.g. 1000 l will be capable of holding at least 1000 l, with the precise maximum working volume being a function of the vessel geometry.

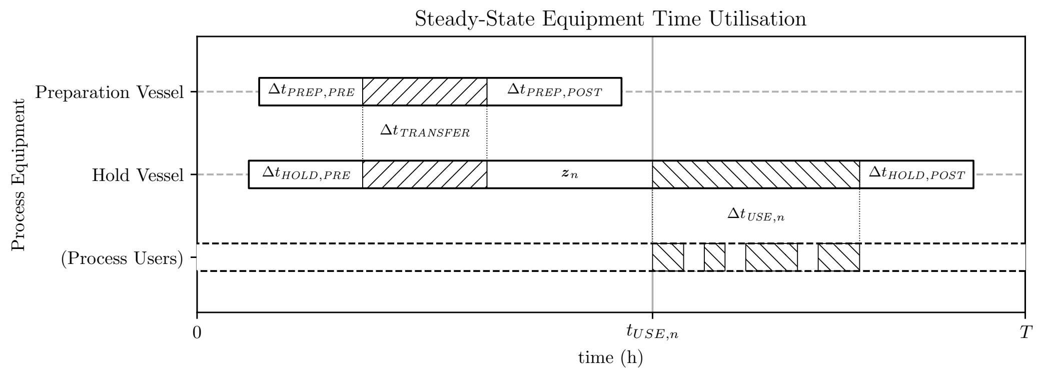


Figure 3.1: Equipment time utilisation for a single buffer

Table 3.2: Buffer data for random example

names	required volumes	use start times	use durations
	U_n (l)	$t^*_{USE,n}$ (h)	$\Delta t_{USE,n}$ (h)
Buffer #1	24 427.13	76.23	20.56
Buffer #2	5487.29	0.21	49.77
Buffer #3	2588.36	25.78	24.56
Buffer #4	7102.05	46.79	27.77
Buffer #5	1020.87	87.7	36.58
Buffer #6	19 508.79	35.52	58.53
Buffer #7	23 073.55	42.26	39.71
Buffer #8	25 454.10	48.38	43.47
Buffer #9	24 088.67	4.18	55.41
Buffer #10	3172.46	48.31	23.27
Buffer #11	24 752.71	76.38	45.80
Buffer #12	13 445.31	73.93	34.25

3.3 Buffer Data

The buffer data will contain several parameters for each buffer to be prepared, $n \in \mathcal{N}$. A buffer data-set will contain entries for N buffers:

$$n \in \mathcal{N}; \quad \mathcal{N} = \{0, 1, 2, \ldots, n, \ldots, (N-1)\} \tag{3.3.1}$$

$$N = |\mathcal{N}| \tag{3.3.2}$$

For each buffer, $n \in \mathcal{N}$, the data set will contain, at a minimum, its required preparation volume (in litres), U_n.

If nothing was known about the scheduling of the production process, a simple simulation could be carried out without scheduling. Such a simulation would assume a fixed duration for all operations in the preparation procedure and would require knowledge of the *cycle time* (the fixed duration from the start of one batch to the start of another batch) and an additional parameter representing the maximum utilisation ratio of the preparation vessels (see Section 3.4).

For a complete model, with scheduling, we also need to know when each buffer is first required by the process, and the duration for which it is required.

Sample buffer data are given in Table 3.2. Figure 3.1 gives an overview of data and parameters related to scheduling, which are detailed in the remainder of this section and in Section 3.4. The plot in Figure 3.1 is termed an *equipment time utilisation* plot; in this instance it just shows the preparation and hold procedures for a single buffer. This style of plot is covered in more detail in Chapter 5, where similar plots can be used to visually display a working steady-state schedule for a feasible solution.

For each buffer, $n \in \mathcal{N}$, its (unnormalised) time of first use, $t^*_{USE,n}$, is the time when the process draws the first drop of buffer from its hold vessel. This time is relative to some batch datum, e.g. the start of the batch or the start of downstream processing. The choice of datum is unimportant once it is consistently used. A campaign is a series of batches. The model deals with *steady-state* processing (batches start with a fixed cycle time and are assumed to be in the middle of a long campaign), so we are interested in a single-cycle window. Since we are looking at a single-cycle window at steady-state, the single-cycle windows either side of it will be identical, and so on until we approach the edges of the long campaign. As a result, we want to normalise $t^*_{USE,n}$ with respect to the cycle time, T. Accordingly, we define the (normalised) time of first use, $t_{USE,n}$:

$$t_{USE,n} = t^*_{USE,n} \mod T \quad \forall n \in \mathcal{N}. \tag{3.3.3}$$

Note that the random data generator used to produce the data in Table 3.2 produces values for $t^*_{USE,n}$ that are already normalised, but for the real-world examples discussed in Chapter 5, $t^*_{USE,n}$ may be unnormalised.

For each buffer, $n \in \mathcal{N}$, its duration of first use, $\Delta t_{USE,n}$, is the duration from when the process draws the first drop of buffer from its hold vessel to when it finishes drawing the last drop of buffer from the same vessel. Note that a process may draw buffer discontinuously from a hold vessel, e.g. a cleaning buffer may be drawn from its hold vessel for a few minutes at the end of a chromatography procedure and may not be required again until near the end of another chromatography procedure a day or two later, but may not be required at all in the intervening period. Note the use of the general convention that *durations* are denoted by Δt, whereas *times* (relative to some datum) are denoted by t. It is assumed that all times and durations are in hours.

For each buffer, $n \in \mathcal{N}$, $\Delta t_{USE,n}$ must be sufficiently less than the cycle time so that there is opportunity to *turn around* its hold vessel (i.e. sufficient time must exist after use of the buffer to clean and sterilise the vessel, receive the subsequent batch of the sane buffer and hold it for a sufficient duration before it is required by the subsequent batch).

3.4 Parameters

In addition to the buffer-specific and vessel-specific data detailed above, some global parameters are required to specify the problem. For the random example, these parameters are tabulated in Table 3.3 and are described in more detail hereafter.

The first parameter specified is the cycle time, T. We are concerned with the steady-state operation of the process, so the cycle time can be thought of as the duration from some fixed point in one batch to the same fixed point in the subsequent batch. The cycle time is typically some multiple of 24 hours. This

Table 3.3: Global parameters for random example

symbol	short description	value	unit
T	process cycle time	96.0	h
$\Delta t_{PREP,PRE}$	prep pre duration	12.0	h
$\Delta t_{PREP,POST}$	prep post duration	1.5	h
$\Delta t_{TRANSFER}$	transfer duration	2.0	h
$\Delta t_{HOLD,PRE}$	hold pre duration	8.0	h
$\Delta t_{HOLD,POST}$	hold post duration	1.5	h
$\Delta t_{HOLD,MIN}$	minimum hold duration	12.0	h
$\Delta t_{HOLD,MAX}$	maximum hold duration	60.0	h
$f_{MINFILL}$	minimum fill ratio	0.3	–
f_{UTIL}	maximum utilisation ratio	0.8	–

is for reasons of operability; it is easier for staff if a given task occurs at the same time of the day for each batch. Note that it is not our objective to seek to minimise the cycle time. A well-designed facility should be bottlenecked at the production bioreactor, so the cycle time is a function of the production process and not the buffer preparation area.

In the model, the preparation and hold procedures are both broken into a number of operations. The durations of most of these operations are specified as global parameters. It is unlikely that detailed timings will be available at the early stages of design, so specifying some conservative global parameters provides a model that is easy to comprehend and validate. In reality, operations such as filling a vessel with water will scale with buffer volume and operations such as cool-down of a vessel after steaming will scale with vessel volume. This degree of granularity could be added as a future model enhancement.

The following durations are specified as global parameters:

- The parameter $\Delta t_{PREP,PRE}$ is the duration of all operations in the preparation procedure prior to the transfer of buffer from the preparation vessel to the hold vessel. This may include pressure testing, steam-in-place (SIP) and cool-down, although these tasks are not always carried out on preparation vessels. The period will include the charging of water for injection (WFI), the charging of other liquids and/or solids, and time for mixing the buffer. It may also include some time for sampling or testing and adjustment of the buffer.

- The parameter $\Delta t_{PREP,POST}$ is the duration of all operations in the preparation procedure that take place after the transfer of the buffer to the buffer hold vessel is complete. This is typically just comprised of the clean-in-place (CIP) of the vessel. CIP involves a separate piece of equipment, the CIP skid, which circulates cleaning fluids (detergents and/or acids, bases) through the vessel and some of its attached pipework.

- The parameter $\Delta t_{TRANSFER}$ is the duration of the transfer of buffer from the preparation vessel to the hold vessel, via a sterile filter. Although this varies with the buffer volume, the relationship is not that straightforward. As buffer and vessel volumes increase, a change in pipe diameter or pump size may be appropriate, giving large step changes to the transfer duration. Accordingly, a conservative global parameter for transfer duration is appropriate for early-stage design.

- The parameter $\Delta t_{HOLD,PRE}$ is the duration of all operations in the hold procedure prior to the commencement of receipt of buffer from the preparation vessel. This typically includes pressure testing, SIP and cooldown.

- The parameter $\Delta t_{HOLD,POST}$ is the duration of all operations in the hold procedure that take place after the last use of buffer by the process is complete. Like $\Delta t_{PREP,POST}$, this is primarily comprised of a CIP of the respective vessel.

- Looking at Figure 3.1, the one symbol not yet mentioned is the buffer hold duration, $\boldsymbol{z}_n$. Note that this is a decision variable, and not a parameter; it is discussed in more detail in Chapter 4. The buffer hold duration is the duration that spans from the end of the transfer into the buffer hold vessel until the start of the first use of the buffer by the process. As will be described in Chapter 4, we place some bounds on this duration and express these bounds as a pair of global parameters. The parameter $\Delta t_{HOLD,MIN}$ defines the minimum allowable duration of the buffer hold operation. In practice, the operators of a plant do not want to schedule buffer preparations so that the transfer is complete moments before the buffer is required by the process, as any delays at this stage could impact production. A value of $\Delta t_{HOLD,MIN}$, typically in the range of several hours to one day, can be specified to ensure that buffers are prepared with some flexibility to handle delays. At the other end of the scale, buffers may expire and the global variable $\Delta t_{HOLD,MAX}$ is defined to set a maximum allowable hold duration.

- When selecting a vessel, a minimum fill ratio, $f_{MINFILL}$ is defined to ensure that a vessel isn't selected that is far too big for the buffer being prepared. The rationale for this was discussed in Section 3.2.

- Finally, the utilisation ratio, f_{UTIL}, places an upper limit on how busy a preparation vessel is allowed to be. In the absence of detailed buffer scheduling information, this may represent an 'engineering factor' to account for unforeseeable scheduling clashes. If detailed scheduling information is available, the ratio can be set to unity to effectively remove its influence as a constraint, or the ratio could be maintained at some value less than one to allow for overhead in the design.

3.5 Data Sources

3.5.1 Random Data

The buffer data in Table 3.2 were randomly generated. The parameters in Table 3.3 and the vessel data in Table 3.1 are typical figures based on the author's experience of designing and optimising bioprocessing facilities.

A function was developed to generate random buffer data. The function requires four inputs; the number of buffers required, $\mathcal{N}$, the minimum duration ratio, f_{MINUSE}, the maximum duration ratio, f_{MAXUSE}, and a 'parameters' data set. The function generates the required number of buffers, each with $t^*_{USE,n}$ uniformly distributed in the range $0 < t^*_{USE,n} < T$. Note that, for the random data, this means that $t^*_{USE,n} = t_{USE,n}$, i.e. the random use times are already normalised. The use durations, $\Delta t_{USE,n}$ are uniformly distributed and depend on several parameters so that the model is feasible. The feasible range may be further scaled back using f_{MINUSE} and f_{MAXUSE} to ensure values are credible, i.e.

$$f_{MINUSE} \times T < \Delta t_{USE,n} < f_{MAXUSE} \times \Delta t_{FEAS} \tag{3.5.1}$$

where

$$\Delta t_{FEAS} = T - (\Delta t_{HOLD,PRE} + \Delta t_{TRANSFER} + \Delta t_{HOLD,MIN} + \Delta t_{HOLD,POST}) \,. \tag{3.5.2}$$

These values are bound below by the minimum duration ratio times the cycle time

3.5.2 Real-World Data

Real-world data sets are based on mass balances and scheduling simulations performed by the author for biopharmaceutical clients. Since these processes are confidential, the data has been obfuscated in several ways. Firstly, the buffers are all given terse names, such as 'Buffer #1', and are in no particular order. Secondly, the buffer volumes may be scaled but only to a degree that does not affect vessel selection. Thirdly, the (normalised) duration parameters may all be scaled by a common factor. These changes serve to make the data unrecognisable but do not affect vessel selection or scheduling (save for scaling the time axis).

Two real-world data sets are included as appendices to this report.

Chapter 4

Methodology

Just as the largest library, badly arranged, is not so useful as a very moderate one that is well arranged, so the greatest amount of knowledge, if not elaborated by our own thoughts, is worth much less than a far smaller volume that has been abundantly and repeatedly thought over. For only by universally combining what we know, by comparing every truth with every other, do we fully assimilate our own knowledge and get it into our power.

— Arthur Schopenhauer, *On Thinking for Oneself*

4.1 Introduction

At first glance, the pathway to solving the vessel selection problem was unclear. There are two aspects to the problem; scheduling and selection.

For the scheduling aspect of the problem, initial concepts involved finding an API to one of the commercial scheduling packages. One initial concept included developing an algorithm or heuristic to intelligently guess at possible vessel selections, and pass these into a scheduling model which could be repeatedly be solved using a commercial scheduling package to find an optimum selection. It became evident that this concept would be cumbersome and computationally difficult.

The vessel selection problem appears to be a close relative of a number of classic **NP**-hard problems in the 'bin-packing' genre so methods for solving such problems were investigated to see if any methodologies could be adapted to suit the problem at hand. It was noted that bin-packing problems, as well as some scheduling problems, such as job-shop scheduling, were solved using mixed-integer linear programming (MILP) – see, e.g. Martello and Toth (1990), Taha (2017).

The focus thus centred on formulating the problem as a MILP problem. Firstly, the problem was described mathematically. Next, attention was focused on the method of solution.

4.2 Slots

To model the problem as an MILP problem, we must introduce the concept of *slots*. Noting that there is a buffer hold vessel for each buffer, it is evident that a feasible but inefficient solution could involve installing a dedicated, suitably sized, preparation vessel for each buffer. Any solution involving more than this number of preparation vessels cannot be optimal. As a result, we have an upper limit of N on the number of required preparation vessels. The model is thus constructed with P *slots*, where

$$P = N. \tag{4.2.1}$$

A slot, $p \in \mathcal{P}$, is a notional space which may either be occupied by a vessel, or remain empty;

$$p \in \mathcal{P}; \quad \mathcal{P} = \{0, 1, 2, \ldots, p, \ldots, (P-1)\} \tag{4.2.2}$$

and

$$P = |\mathcal{P}|. \tag{4.2.3}$$

Note that slots do not have any physical significance, i.e. in a real-world facility, floor space is assigned to physical vessels, rather than notional slots.

4.3 Objective Function

The vessel selection problem may be described as a series of linear constraints. These constraints are applied to find the optimum value of an objective function, which we seek to minimise. The primary objective function is the total cost of vessels. We find this by summing the vessel costs for all vessels present in slots. Recall that the vessel data set contains entries for $\mathcal{M}$ vessel sizes, each of which has an associated cost, c_m.

We now introduce the first decision variable, $\boldsymbol{y}_{mp}$. This a binary decision variable with dimensions $M \times P$. Note that binary decision variables may only take the values 0 and 1, i.e.

$$\boldsymbol{y}_{mp} \in \{0, 1\} \quad \forall m \in \mathcal{M} \quad \forall p \in \mathcal{P}. \tag{4.3.1}$$

The possible states of $\boldsymbol{y}_{mp}$ may be described thus:

$$\boldsymbol{y}_{mp} = \begin{cases} 1 \implies \text{instance of vessel } m \text{ in slot } p \\ 0 \implies \text{otherwise.} \end{cases} \tag{4.3.2}$$

We can now describe the objective function, which gives an expression for the objective function *value*, $\boldsymbol{Z}$. The primary objective is to minimise $\boldsymbol{Z}$.

$$\boldsymbol{Z} = \sum_{m \in \mathcal{M}} \sum_{p \in \mathcal{P}} c_m \boldsymbol{y}_{mp}, \quad \boldsymbol{Z} \in \mathbb{R}; \ \boldsymbol{Z} \geq 0. \tag{4.3.3}$$

4.4 Basic Model

A small number of constraints need to be applied to arrive at the simplest variant of the problem. Additional constraints may then be added to make the model more detailed or realistic. The basic model requires no information on when each buffer is required by the process and only provides a rough estimation of vessel requirements as a result. The complete model, detailed in Section 4.5, exists as an extension of the basic model, with the introduction of scheduling constraints.

4.4.1 Buffers Dedicated to Slots

The first constraint to be added is the limitation that a buffer must be prepared in exactly one slot. Recall that we process one batch per cycle. This constraint means that we must prepare each buffer once per cycle and also that the buffer is always prepared in the same slot, i.e. slot selection is identical for every cycle. We introduce a new binary decision variable, $\boldsymbol{x}_{np}$, which takes a value of 1 if buffer n is prepared in slot p, and takes a value of 0 otherwise, i.e.

$$\boldsymbol{x}_{np} \in \{0, 1\} \quad \forall n \in \mathcal{N} \quad \forall p \in \mathcal{P}. \tag{4.4.1}$$

Note that $\boldsymbol{x}_{np}$ has dimensions of $N \times P$. The possible states of $\boldsymbol{x}_{np}$ may be described thus:

$$\boldsymbol{x}_{np} = \begin{cases} 1 \implies \text{buffer } n \text{ prepared in slot } p \\ 0 \implies \text{otherwise.} \end{cases} \tag{4.4.2}$$

The following constraint can now be defined:

$$\sum_{p \in \mathcal{P}} \boldsymbol{x}_{np} = 1 \quad \forall n \in \mathcal{N} \tag{4.4.3}$$

4.4.2 Vessel Instances Dedicated to Slots

The second constraint to be added is the requirement that at most one vessel may inhabit a given slot. This is not directly analogous to the first constraint; it is possible to use the same *sized* vessel in many slots, but a maximum of one vessel *instance* may inhabit any given slot. Note that this inequality allows for the possibility of unused slots, i.e. the number of occupied slots (and hence the number of preparation vessels) may be less than the number of available slots (and hence the number of buffers).

$$\sum_{m \in \mathcal{M}} \boldsymbol{y}_{mp} \leq 1 \quad \forall p \in \mathcal{P} \tag{4.4.4}$$

4.4.3 Vessel Capacity

The third constraint is the requirement that, if a vessel is in a given slot, it has an appropriate volume to prepare all buffers assigned to the slot. There are two facets to this requirement. Firstly, we cannot produce buffer volumes greater than the preparation vessel maximum working volume. Additionally, vessels have a minimum fill ratio, $f_{MINFILL}$, usually in the range 10 % to 30 %. The minimum fill ratio is generally a function of the minimum stir volume of a vessel's impeller; adequate mixing cannot be guaranteed if the volume is too low.

More complicated rules can be envisaged, such as the ability to make an excess of buffer to prevent the requirement for adding a smaller vessel, discarding the excess in each batch. Such approaches are beyond the scope of this basic model.

Recall that for each vessel size, $m \in \mathcal{M}$, we have defined a maximum vessel working volume, V_m. Also, for each buffer, $n \in \mathcal{N}$, we have defined the required volume, U_n.

The maximum vessel capacity constraint is given by:

$$U_n \boldsymbol{x}_{np} \leq \sum_{m \in \mathcal{M}} V_m \boldsymbol{y}_{mp} \quad \forall n \in \mathcal{N}, \ \ \forall p \in \mathcal{P} \tag{4.4.5}$$

Note that the summation on the right hand side of equation 4.4.5 will contain *at most* one non-zero term. If there is no vessel in slot p, then $\boldsymbol{y}_{mp} = 0 \ \ \forall m \in \mathcal{M}$. If there is a vessel of size m in slot p, then $\boldsymbol{y}_{mp} = 1$. Thus, the summation term is used to select the volume of the vessel (if any) in slot p.

The minimum vessel capacity constraint is slightly more complex, in that the *big-M* method must be utilised. The *big-M* method is explained in detail in e.g. Chapter 3 of Taha (2017) and is briefly explained by example here.

Imagine we have a constraint involving two positive, real variables, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, e.g. $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$, and we only want to apply the constraint if a particular binary variable, $\boldsymbol{\gamma}$, has a value of one. We introduce a new constant, $\mathbb{M}$, which is a *sufficiently large* number and rewrite the constraint as:

$$\mathbb{M} + \boldsymbol{\alpha} \geq \boldsymbol{\beta} + \mathbb{M}\boldsymbol{\gamma} \tag{4.4.6}$$

In the above equation, if $\boldsymbol{\gamma} = 0$, then $\boldsymbol{\beta} \leq \boldsymbol{\alpha} + \mathbb{M}$. This is always true since we have chosen a sufficiently large value for $\mathbb{M}$, i.e. the constraint is *deactivated* when $\boldsymbol{\gamma} = 0$. Conversely, if $\boldsymbol{\gamma} = 1$, the two terms containing $\mathbb{M}$ cancel and we are left with the original constraint, $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$, i.e. the original constraint is *activated* when $\boldsymbol{\gamma} = 1$. By inspection, a suitably large value for $\mathbb{M}$ in the above would be $\mathbb{M} = \max(\boldsymbol{\beta} - \boldsymbol{\alpha})$.

Returning to the original problem, we set $\mathbb{M} = V_{MAX}$, where:

$$V_{MAX} = \max(V_m \ \ \forall m \in \mathcal{M}) \tag{4.4.7}$$

The minimum vessel capacity constraint is then defined as:

$$V_{MAX} + U_n \geq f_{MINFILL} \sum_{m \in M} V_m \boldsymbol{y}_{mp} + V_{MAX} \boldsymbol{x}_{np} \quad \forall n \in \mathcal{N}, \ \forall p \in \mathcal{P} \tag{4.4.8}$$

4.4.4 Preparation Vessel Utilisation

In the absence of detailed scheduling data (i.e. knowledge of when the buffers are used by the process and their durations of use), an allowance can be made for unknown scheduling constraints by limiting the maximum allowed utilisation of the preparation vessels. A slot is deemed to be utilised when a preparation procedure is occurring therein. A slot's utilisation ratio is the total duration of all such utilisation in a cycle, expressed as a fraction of the cycle time.

We thus introduce the parameter f_{UTIL}, the maximum utilisation ratio. The total duration of all preparation procedures in a given slot must not be above $f_{UTIL} \times T$, where T is the cycle time of the process.

As can be seen from Figure 3.1, the total duration of a single preparation procedure, Δt_{PREP}, is given by:

$$\Delta t_{PREP} = \Delta t_{PREP,PRE} + \Delta t_{TRANSFER} + \Delta t_{PREP,POST} \tag{4.4.9}$$

Note that, for the basic case, it is only strictly necessary to estimate the preparation procedure duration, Δt_{PREP}; the constituent operation durations do not need to be known individually.

The following constraint is now defined:

$$\Delta t_{PREP} \sum_{n \in \mathcal{N}} \boldsymbol{x}_{np} \leq f_{UTIL} T \quad \forall p \in \mathcal{P} \tag{4.4.10}$$

Note that the application of a maximum utilisation constraint in the absence of scheduling data will result in a crude approximation but it is often the case that detailed timing information is not available or cannot be predicted with sufficient accuracy. Setting a conservatively low value for f_{UTIL} is essentially an assertion that, in the absence of scheduling data, if all slots are utilised less than f_{UTIL} of the time, it is probable that the selected vessels are sufficient to schedule the process without clashes. It should be apparent that this is essentially a fudge, but at the early stage of a design process and in the absence of better data, it may be sufficient for a first guess at the vessel requirements and may be sufficient for a rough feasibility-stage cost estimate. It is thus apparent that accurate scheduling data are required if a more accurate result is necessary. The application of scheduling constraints forms the basis of the *complete* model, detailed in Section 4.5.

4.4.5 Basic Model Summary

The basic model is summarised below. The constraint equations have been rearranged so that variables are on the left hand side and constants are on the right.

Minimise:

$$\boldsymbol{Z} = \sum_{m \in \mathcal{M}} \sum_{p \in \mathcal{P}} c_m \boldsymbol{y}_{mp} \tag{4.3.3}$$

Subject to:

$$\sum_{p \in \mathcal{P}} \boldsymbol{x}_{np} = 1 \quad \forall n \in \mathcal{N} \tag{4.4.3}$$

$$\sum_{m \in \mathcal{M}} \boldsymbol{y}_{mp} \leq 1 \quad \forall p \in \mathcal{P} \tag{4.4.4}$$

$$U_n \boldsymbol{x}_{np} - \sum_{m \in \mathcal{M}} V_m \boldsymbol{y}_{mp} \leq 0 \quad \forall n \in \mathcal{N}, \ \forall p \in \mathcal{P} \tag{4.4.5}$$

$$V_{MAX} \boldsymbol{x}_{np} + f_{MINFILL} \sum_{m \in \mathcal{M}} V_m \boldsymbol{y}_{mp} \leq U_n + V_{MAX} \quad \forall n \in \mathcal{N}, \ \forall p \in \mathcal{P} \tag{4.4.8}$$

$$\Delta t_{PREP} \sum_{n \in \mathcal{N}} \boldsymbol{x}_{np} \leq f_{UTIL} T \quad \forall p \in \mathcal{P} \tag{4.4.10}$$

Where:

$$\boldsymbol{x}_{np} \in \{0, 1\} \quad \forall n \in \mathcal{N}, \ \forall p \in \mathcal{P} \tag{4.4.1}$$

$$\boldsymbol{y}_{mp} \in \{0, 1\} \quad \forall m \in \mathcal{M}, \ \forall p \in \mathcal{P} \tag{4.3.1}$$

4.5 Complete Model

For a more accurate appraisal of vessel requirements, scheduling data are required for each buffer. Specifically, we require the duration of use of each buffer, along with the time of first use of each buffer. Given this information, it is possible to constrain the problem so that the individual preparations are scheduled correctly.

For each buffer, constraints covering both the scheduling of the preparation procedures and the scheduling of the hold procedures must be added. The scheduling of the hold procedures is quite straightforward; since the hold vessels are dedicated, we simply need to ensure that, for each buffer, the total duration of its hold procedure is not greater than the cycle time. The scheduling of the preparation procedures is more complex, despite these procedures being of fixed duration.

4.5.1 Hold Procedure Duration

We now introduce the buffer hold duration decision variable, $\boldsymbol{z}_n$:

$$\Delta t_{HOLD,MIN} \leq \boldsymbol{z}_n \leq \Delta t_{HOLD,MAX}; \quad \boldsymbol{z}_n \in \mathbb{R} \quad \forall n \in \mathcal{N} \tag{4.5.1}$$

This is the only decision variable in the model that is not a binary variable.

The first constraint required for the complete model is the limitation that the total duration of each hold procedure must not be greater than the cycle time.

If this constraint is not observed, a hold procedure in a given batch may not have finished before the hold procedure for the next batch is due to start.

$$\boldsymbol{z}_n \leq T - (\Delta t_{HOLD,PRE} + \Delta t_{TRANSFER} + \Delta t_{USE,n} + \Delta t_{HOLD,POST}) \quad \forall n \in \mathcal{N} \tag{4.5.2}$$

4.5.2 Introducing the Scheduling Constraint

The buffer preparation scheduling constraint may be described quite simply; *Two events requiring the same piece of equipment must not occur simultaneously.*

Since we have a constant preparation duration, Δt_{PREP}, this constraint may be expressed, *for any two distinct buffers that are made in the same slot*, by the following equation, which is only valid for *non-cyclic* operation:

$$|\boldsymbol{t}_{PREP,k} - \boldsymbol{t}_{PREP,n}| \geq \Delta t_{PREP} \quad \forall n \in \mathcal{N}, \ \forall k \in \mathcal{N}; \ k > n \tag{4.5.3}$$

where $\boldsymbol{t}_{PREP,k}$ and $\boldsymbol{t}_{PREP,n}$ are the *reference times* for the preparation procedures of buffers k and n respectively. Note that the range of k is limited to $k > n$ to prevent duplication of constraints. The above formula is not yet in a format that can be applied in a MILP model.

Firstly, it was noted that the constraints only apply to two buffers which happen to be made in the same slot; implementing this requirement will necessitate some additional logic.

Secondly, absolute value expressions are not valid in linear programming constraints; converting to valid expressions will also require some additional logic.

Thirdly, we haven't yet defined $\boldsymbol{t}_{PREP,k}$ and $\boldsymbol{t}_{PREP,n}$; they represent the reference times of preparation procedures k and n in the current cycle window and will be described in more detail in Section 4.5.5.

Finally, the above expression isn't always a valid representation of the constraint when dealing with a cyclic process. For example, if $\boldsymbol{t}_{PREP,k}$ has a value just below T and $\boldsymbol{t}_{PREP,n}$ has a value just above zero, the two preparation procedures may clash due to them crossing the cycle boundary; some additional logic is required to ensure such boundary cases are treated properly.

To overcome the issues outlined above, several additional constraints and variables must be introduced.

4.5.3 Pairs of Distinct Buffers Prepared in the Same Slot

We want to specify a binary variable which indicates if two distinct buffers are prepared in the same slot. This, in turn requires an additional binary variable which indicates if two distinct buffers are prepared in a *particular* slot. The latter binary variable, $\boldsymbol{w}_{nkp}$, is defined as follows:

$$\boldsymbol{w}_{nkp} \in \{0,1\} \quad \forall n \in \mathcal{N},\ \forall k \in \mathcal{N};\ k > n,\ \forall p \in \mathcal{P} \tag{4.5.4}$$

$$\begin{aligned} \boldsymbol{w}_{nkp} &\leq \tfrac{1}{2}\boldsymbol{x}_{np} + \tfrac{1}{2}\boldsymbol{x}_{kp} - 1 \\ \boldsymbol{w}_{nkp} &\geq \ \boldsymbol{x}_{np} + \ \boldsymbol{x}_{kp} - 1 \end{aligned} \quad \forall n \in \mathcal{N},\ \forall k \in \mathcal{N};\ k > n,\ \forall p \in \mathcal{P} \tag{4.5.5}$$

Note that $\boldsymbol{w}_{nkp}$ has dimensions $\binom{N}{2} \times P$. Since $\boldsymbol{w}_{nnp}$ is not required (we do not need to compare a buffer with itself) and $\boldsymbol{w}_{nkp} = \boldsymbol{w}_{knp}$, we only need to model constraints where $k > n$.

A truth table for the constraints in equation 4.5.5 is given in Table 4.1. In this table, $\boldsymbol{w}^{(1)}_{nkp}$ refers to the first inequality above and $\boldsymbol{w}^{(2)}_{nkp}$ refers to the second. Applying both inequalities is equivalent to performing a logical-and, i.e. $\boldsymbol{w}_{nkp} = \boldsymbol{w}^{(1)}_{nkp} \wedge \boldsymbol{w}^{(2)}_{nkp}$. It can be seen from Table 4.1 that $\boldsymbol{w}_{nkp}$ is only equal to one when both $\boldsymbol{x}_{np}$ and $\boldsymbol{x}_{np}$ are equal to one, i.e. $\boldsymbol{w}_{nkp} = 1$ iff buffer k and buffer n are both prepared in slot p.

Table 4.1: Truth table for $\boldsymbol{w}_{nkp}$

$\boldsymbol{x}_{np}$	$\boldsymbol{x}_{kp}$	$\boldsymbol{w}^{(1)}_{nkp}$	$\boldsymbol{w}^{(2)}_{nkp}$	$\boldsymbol{w}_{nkp} = \boldsymbol{w}^{(1)}_{nkp} \wedge \boldsymbol{w}^{(2)}_{nkp}$
0	0	0	$\{0,1\}$	0
0	1	0	$\{0,1\}$	0
1	0	0	$\{0,1\}$	0
1	1	$\{0,1\}$	1	1

We can now define a binary variable, $\boldsymbol{a}_{nk}$ which indicates if two distinct buffers are prepared in the *same* slot.

$$\boldsymbol{a}_{nk} \in \{0, 1\} \quad \forall n \in \mathcal{N},\ \forall k \in \mathcal{N};\ k > n \tag{4.5.6}$$

This new binary variable is introduced via the following constraint:

$$\boldsymbol{a}_{nk} = \sum_{p \in \mathcal{P}} \boldsymbol{w}_{nkp} \quad \forall n \in \mathcal{N},\ \forall k \in \mathcal{N};\ k > n \tag{4.5.7}$$

Note that we may use $\boldsymbol{a}_{nk}$ in describing some subsequent equations, but since it is a strict equality, $\boldsymbol{a}_{nk}$ may be replaced by the right hand side of equation 4.5.7 in the final model to minimise model complexity. Neglecting to do so would give rise to the unnecessary addition of $\binom{N}{2}$ variables and $\binom{N}{2}$ constraints.
A truth table for $\boldsymbol{a}_{nk}$ is given in Table 4.2. When $\boldsymbol{a}_{nk} = 1$, buffers n and k are prepared in the same vessel so we are interested in preventing their preparation procedures from clashing. If $\boldsymbol{a}_{nk} = 0$, the buffers are not prepared in the same vessel, so we don't need to constrain how they are scheduled relative to one another.

Table 4.2: Truth table for $\boldsymbol{a}_{nk}$

$\boldsymbol{w}_{nk0}$	$\boldsymbol{w}_{nk1}$	$\cdots$	$\boldsymbol{w}_{nk(P-1)}$	$\boldsymbol{w}_{nkP}$	$\boldsymbol{a}_{nk}$
0	0	$\cdots$	0	0	0
0	0	$\cdots$	0	1	0
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
1	1	$\cdots$	1	0	0
1	1	$\cdots$	1	1	1

4.5.4 Absolute value constraints

We still cannot apply the scheduling constraint introduced in equation 4.5.3 due to the presence of the absolute value expression. Given two events, both of (fixed) duration δ, with respective (variable) start times $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, the absolute value expression may be though of as representing a pair of constraints, e.g. $|\boldsymbol{\beta} - \boldsymbol{\alpha}| \geq \delta$ is essentially shorthand for the expression

$$\boldsymbol{\beta} - \boldsymbol{\alpha} = \begin{cases} \geq 0 \implies \boldsymbol{\beta} - \boldsymbol{\alpha} \geq \delta \\ \leq 0 \implies \boldsymbol{\beta} - \boldsymbol{\alpha} \leq -\delta \end{cases} \tag{4.5.8}$$

where $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$ are positive, real variables and δ is a positive, real constant. The pair of expressions in equation 4.5.8 represent an *or* constraint, which must be converted to an *and* constraint if they are to be included in a MILP model. This conversion requires the definition of an additional binary variable, $\boldsymbol{\epsilon}$, which is used to select between the two cases in equation 4.5.8. The *big-M* method (see Section 4.4.3) is used to arrive at correct equations suitable for a MILP model:

$$\begin{aligned} \mathbb{M}\boldsymbol{\epsilon} + \boldsymbol{\beta} - \boldsymbol{\alpha} &\geq \delta \\ \mathbb{M}\boldsymbol{\epsilon} + \boldsymbol{\beta} - \boldsymbol{\alpha} &\leq -\delta + \mathbb{M} \end{aligned} \tag{4.5.9}$$

Note that the binary variable $\boldsymbol{\epsilon}$ is not constrained by any other equations, i.e. $\boldsymbol{\epsilon}$ is a free variable that allows for selection between the two constraints in equation 4.5.9. Also note that $\mathbb{M}$ is a *suitably large* number, e.g. by inspection, $\mathbb{M} = \max\left(|\boldsymbol{\beta} - \boldsymbol{\alpha}| + \delta\right)$ is sufficient.

A truth table for the possible states of $\boldsymbol{\beta} - \boldsymbol{\alpha}$ is given in equation 4.3. Note that the case $\boldsymbol{\beta} - \boldsymbol{\alpha} = 0$ can only occur if $\delta = 0$ since we have defined δ to be positive. This means that the two events can only occur at the same time if they have zero duration; a trivial edge case which will not arise in the proposed model.

Table 4.3: Truth table for $\boldsymbol{\beta} - \boldsymbol{\alpha}$

$\boldsymbol{\beta} - \boldsymbol{\alpha}$	$\boldsymbol{\epsilon}$	active constraint
< 0	1	$\boldsymbol{\beta} - \boldsymbol{\alpha} \geq \delta$
0	$\{0, 1\}$	$\delta \leq 0$
> 0	0	$\boldsymbol{\beta} - \boldsymbol{\alpha} \leq -\delta$

4.5.5 Preparation Procedure Reference Times

Recall that in Section 4.5.2, the preparation procedure reference times, $\boldsymbol{t}_{PREP,k}$ and $\boldsymbol{t}_{PREP,n}$, were introduced without a full definition. For a buffer, n, the variable $\boldsymbol{t}_{PREP,n}$ is defined as:

$$\boldsymbol{t}_{PREP,n} = t_{USE,n} - \boldsymbol{z}_n + T\boldsymbol{q}_n \quad \forall n \in \mathcal{N} \tag{4.5.10}$$

where:

$$0 \leq \boldsymbol{t}_{PREP,n} \leq T, \quad \boldsymbol{t}_{PREP,n} \in \mathbb{R}, \ \forall n \in \mathcal{N} \tag{4.5.11}$$

and:

$$\boldsymbol{q}_n \in \{0,1\} \quad \forall n \in \mathcal{N} \tag{4.5.12}$$

Note that the above constraint is an equality and so, while $\boldsymbol{t}_{PREP,n}$ may be used in equations below, ultimately the right hand side of equation 4.5.10 will be substituted for $\boldsymbol{t}_{PREP,n}$ in the final model equations to minimise the size of the problem.

Equation 4.5.10 contains a new binary variable, $\boldsymbol{q}_n$. The binary variable $\boldsymbol{q}_n$ is used to ensure that $\boldsymbol{t}_{PREP,n}$ lies within the single-cycle range, i.e. $0 \leq \boldsymbol{t}_{PREP,n} \leq T$ via the following constraints:

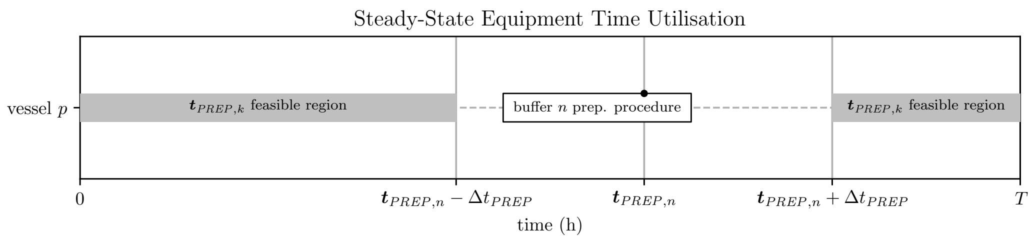
$$\begin{aligned} T\boldsymbol{q}_n + t_{USE,n} - \boldsymbol{z}_n &\geq 0 \\ T\boldsymbol{q}_n + t_{USE,n} - \boldsymbol{z}_n &\leq T \end{aligned} \quad \forall n \in \mathcal{N} \tag{4.5.13}$$

A truth table for $\boldsymbol{q}_n$ is given in Table 4.4. Note that $\boldsymbol{q}_n^{(1)}$ and $\boldsymbol{q}_n^{(2)}$ are the values of $\boldsymbol{q}_n$ in the first and second inequalities in equation 4.5.13 respectively. Applying both inequalities gives $\boldsymbol{q}_n = \boldsymbol{q}_n^{(1)} \wedge \boldsymbol{q}_n^{(2)}$.

Table 4.4: Truth table for $\boldsymbol{q}_n$

$t_{USE,n} - \boldsymbol{z}_n$	$\boldsymbol{q}_n^{(1)}$	$\boldsymbol{q}_n^{(2)}$	$\boldsymbol{q}_n = \boldsymbol{q}_n^{(1)} \wedge \boldsymbol{q}_n^{(2)}$
< 0	1	$\{0,1\}$	1
0	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$
> 0	$\{0,1\}$	0	0

Note that $\boldsymbol{t}_{PREP,n}$ is actually the time, in the current cycle, when the hold operation of buffer n commences (i.e. it is the time, in the current cycle, when



Steady-State Equipment Time Utilisation
vessel p
$\boldsymbol{t}_{PREP,k}$ feasible region
buffer n prep. procedure
$\boldsymbol{t}_{PREP,k}$ feasible region
0
$\boldsymbol{t}_{PREP,n} - \Delta t_{PREP}$
$\boldsymbol{t}_{PREP,n}$
$\boldsymbol{t}_{PREP,n} + \Delta t_{PREP}$
T
time (h)

Figure 4.1: Example feasible region for buffer k scheduling

the transfer of buffer from the preparation vessel to the hold vessel ends. It may appear more intuitive to define $\boldsymbol{t}_{PREP,n}$ as the start time of the preparation procedure for buffer n, but this would lead to a longer expression in equation 4.5.10. Since all preparation procedures have the same duration and we are only ever interested in the difference between pairs of reference times, the definition of $\boldsymbol{t}_{PREP,n}$ given in equation 4.5.10 remains the most concise.

4.5.6 Dealing with Cyclic Time Windows

Imagine, for instance, that a preparation procedure for buffer n occurs in a given slot such that it is scheduled well away from the cycle time boundaries. In such a case, we would envisage two feasible time spans wherein a second preparation procedure (for, e.g. buffer k) may be scheduled in the same slot. In terms of the variables defined in Section 4.5.2, the situation described above is represented graphically in Figure 4.1. The feasible regions may be summarised as follows:

$$\begin{aligned} \boldsymbol{t}_{PREP,k} \geq \boldsymbol{t}_{PREP,n} + \Delta t_{PREP} \quad \text{or} \quad \boldsymbol{t}_{PREP,k} \leq \boldsymbol{t}_{PREP,n} - \Delta t_{PREP} \\ \forall n \in \mathcal{N},\ \ \forall k \in \mathcal{N};\ k > n \end{aligned} \tag{4.5.14}$$

Similar to the approach taken in Section 4.5.4, we can convert the *or* constraints into *and* constraints by defining an additional binary variable, $\boldsymbol{v}_{nk}$ and using the *big-M* method. In this case, $\mathbb{M} = 2T$ is sufficient.

$$\boldsymbol{v}_{nk} \in \{0,1\} \quad \forall n \in \mathcal{N},\ \ \forall k \in \mathcal{N};\ k > n \tag{4.5.15}$$

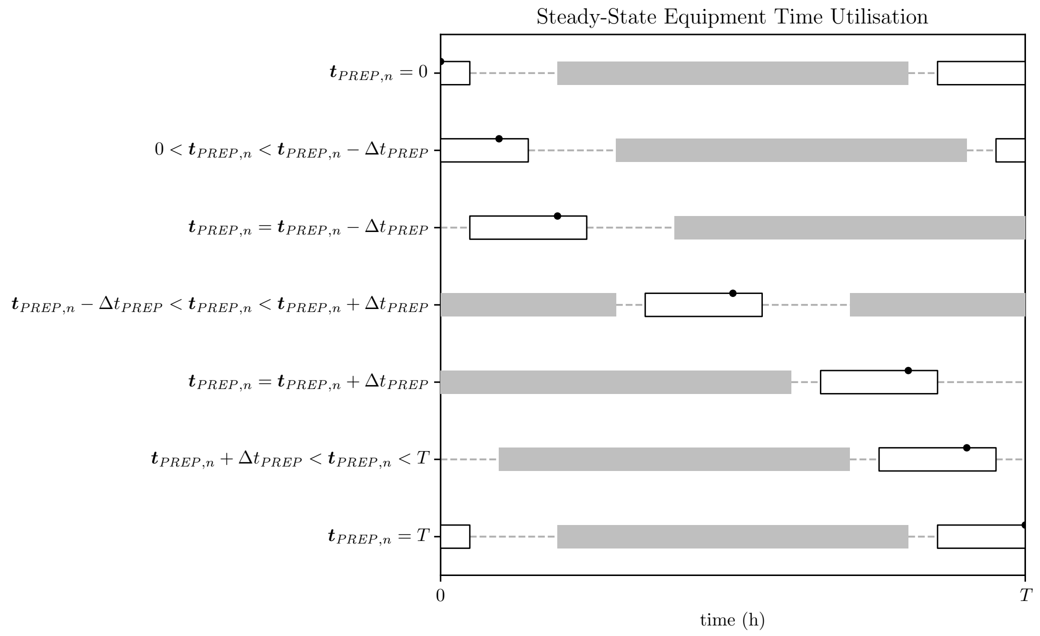


Figure 4.2: All feasible region cases for buffer k scheduling

$$
\begin{aligned}
2T\boldsymbol{v}_{nk} + \boldsymbol{t}_{PREP,k} &\geq \boldsymbol{t}_{PREP,n} + \Delta t_{PREP} \\
2T\boldsymbol{v}_{nk} + \boldsymbol{t}_{PREP,k} &\leq \boldsymbol{t}_{PREP,n} - \Delta t_{PREP} + 2T
\end{aligned}
\qquad \forall n \in \mathcal{N},\ \ \forall k \in \mathcal{N};\ k > n
\tag{4.5.16}
$$

Unfortunately, the above equations do not always hold when the preparation of one or other of the buffers nears a cycle boundary. To explain this, Figure 4.2 (using Figure 4.1 as a legend) explores all the possible cases that must be considered for cyclic scheduling. Note that the case shown in Figure 4.1, i.e. the case where $(\boldsymbol{t}_{PREP,n} - \Delta t_{PREP}) \leq \boldsymbol{t}_{PREP,k} \leq (\boldsymbol{t}_{PREP,n} + \Delta t_{PREP})$ is, in fact, the only case where there are two distinct feasible regions. For all other cases, there is a single feasible region. We need to be able to distinguish between these cases. Note also that the upper and lower bounds of the feasible region cannot always be found by adding or subtracting a factor of Δt_{PREP} to/from $\boldsymbol{t}_{PREP,n}$ respectively, as doing so may result in values outside the single-cycle range in some instances; we need to add some logic to prevent this.

The lower bound of the feasible region, $\boldsymbol{t}_{LOWER,n}$, is defined as:

$$\boldsymbol{t}_{LOWER,n} = \boldsymbol{t}_{PREP,n} + \Delta t_{PREP} - T\boldsymbol{r}_n \quad \forall n \in \mathcal{N} \tag{4.5.17}$$

where:

$$0 \leq \boldsymbol{t}_{LOWER,n} \leq T, \quad \boldsymbol{t}_{LOWER,n} \in \mathbb{R}, \ \forall n \in \mathcal{N} \tag{4.5.18}$$

and:

$$\boldsymbol{r}_n \in \{0, 1\} \quad \forall n \in \mathcal{N} \tag{4.5.19}$$

Similarly, the upper bound of the feasible region, $\boldsymbol{t}_{UPPER,n}$, is defined as:

$$\boldsymbol{t}_{UPPER,n} = \boldsymbol{t}_{PREP,n} - \Delta t_{PREP} + T\boldsymbol{s}_n \quad \forall n \in \mathcal{N} \tag{4.5.20}$$

where:

$$0 \leq \boldsymbol{t}_{UPPER,n} \leq T, \quad \boldsymbol{t}_{LOWER,n} \in \mathbb{R}, \ \forall n \in \mathcal{N} \tag{4.5.21}$$

and:

$$\boldsymbol{s}_n \in \{0, 1\} \quad \forall n \in \mathcal{N} \tag{4.5.22}$$

As was the case with equation 4.5.10, equations 4.5.17 and 4.5.20 are equalities and so $\boldsymbol{t}_{LOWER,n}$ and $\boldsymbol{t}_{UPPER,n}$ will be replaced by the right hand sides of equations 4.5.17 and 4.5.20 respectively in the final model to reduce complexity. The new binary variables, $\boldsymbol{r}_n$ and $\boldsymbol{s}_n$, are used to ensure values are kept in the single-cycle range, similar to the function of $\boldsymbol{q}_n$ in equation 4.5.10:

$$\begin{aligned} T\boldsymbol{r}_n &\leq \Delta t_{PREP} + \boldsymbol{t}_{PREP,n} \\ T\boldsymbol{r}_n &\geq \Delta t_{PREP} + \boldsymbol{t}_{PREP,n} - T \end{aligned} \quad \forall n \in \mathcal{N} \tag{4.5.23}$$

$$\begin{aligned} T\boldsymbol{s}_n &\geq \Delta t_{PREP} - \boldsymbol{t}_{PREP,n} \\ T\boldsymbol{s}_n &\leq \Delta t_{PREP} - \boldsymbol{t}_{PREP,n} + T \end{aligned} \quad \forall n \in \mathcal{N} \tag{4.5.24}$$

Truth tables for equations 4.5.23 and 4.5.24 are given in Tables 4.5 and 4.6 respectively.

As can be seen from Figure 4.2, the feasible region, when contiguous in the single-cycle window, is bounded below by $\boldsymbol{t}_{LOWER,n}$ and above by $\boldsymbol{t}_{UPPER,n}$, giving rise to the following two constraints:

$$\begin{aligned} \boldsymbol{t}_{PREP,k} &\geq \boldsymbol{t}_{LOWER,n} \\ \boldsymbol{t}_{PREP,k} &\leq \boldsymbol{t}_{UPPER,n} \end{aligned} \quad \forall n \in \mathcal{N}, \ \forall k \in \mathcal{N}; \ k > n \tag{4.5.25}$$

Table 4.5: Truth table for $\boldsymbol{r}_n$

$t_{USE,n} + \Delta t_{PREP}$	$\boldsymbol{r}_n^{(1)}$	$\boldsymbol{r}_n^{(2)}$	$\boldsymbol{r}_n = \boldsymbol{r}_n^{(1)} \wedge \boldsymbol{r}_n^{(2)}$
$< T$	0	$\{0,1\}$	0
T	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$
$> T$	$\{0,1\}$	1	1

Table 4.6: Truth table for $\boldsymbol{s}_n$

$t_{USE,n} - \Delta t_{PREP}$	$\boldsymbol{s}_n^{(1)}$	$\boldsymbol{s}_n^{(2)}$	$\boldsymbol{s}_n = \boldsymbol{s}_n^{(1)} \wedge \boldsymbol{s}_n^{(2)}$
< 0	1	$\{0,1\}$	1
0	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$
> 0	$\{0,1\}$	0	0

Recall that constraints on $\boldsymbol{t}_{PREP,k}$ were defined in equation 4.5.16, for the case where there were two distinct feasible regions in the cycle. In equation 4.5.25 above, we have now detailed constraints on $\boldsymbol{t}_{PREP,k}$ for the cases where there is a single, contiguous feasible region in the cycle. All that is left to do is to define another binary variable that selects between these two sets of constraints. By looking at Figure 4.2, it can be seen that we are only interested in applying the constraints in equation 4.5.16 when both $\boldsymbol{t}_{UPPER,n} < \boldsymbol{t}_{PREP,n}$ and $\boldsymbol{t}_{LOWER,n} > \boldsymbol{t}_{PREP,n}$, i.e. when both $\boldsymbol{r}_n = 0$ and $\boldsymbol{s}_n = 0$. In all other cases, we wish to apply the constraints in equation 4.5.25. We therefore define a new binary variable, $\boldsymbol{u}_n$, where $\boldsymbol{u}_n = 0$ iff $\boldsymbol{r}_n = 0$ and $\boldsymbol{s}_n = 0$.

$$\boldsymbol{u}_n \in \{0,1\} \quad \forall n \in \mathcal{N} \tag{4.5.26}$$

$$\begin{aligned} \boldsymbol{u}_n &\leq \boldsymbol{r}_n + \boldsymbol{s}_n \\ \boldsymbol{u}_n &\geq \tfrac{1}{2}\boldsymbol{r}_n + \tfrac{1}{2}\boldsymbol{s}_n \end{aligned} \quad \forall n \in \mathcal{N} \tag{4.5.27}$$

A truth table for equation 4.5.27 is given in Table 4.7. With the definition of $\boldsymbol{u}_n$, a unified set of inequalities can be written which describe the scheduling of buffers prepared in the same slot. Once again, the *big-M* method is employed,

Table 4.7: Truth table for $\boldsymbol{u}_n$

$\boldsymbol{r}_n$	$\boldsymbol{s}_n$	$\boldsymbol{u}_n^{(1)}$	$\boldsymbol{u}_n^{(2)}$	$\boldsymbol{u}_n = \boldsymbol{u}_n^{(1)} \wedge \boldsymbol{u}_n^{(2)}$
0	0	0	$\{0,1\}$	0
0	1	$\{0,1\}$	1	1
1	0	$\{0,1\}$	1	1
1	1	$\{0,1\}$	1	1

with $\mathbb{M} = 2T$.

$$\begin{aligned}
\boldsymbol{t}_{PREP,k} &\leq \boldsymbol{t}_{PREP,n} - \Delta t_{PREP} + \ 2T\boldsymbol{u}_n - 2T\boldsymbol{v}_{nk} + 2T \\
\boldsymbol{t}_{PREP,k} &\geq \boldsymbol{t}_{PREP,n} + \Delta t_{PREP} - \ 2T\boldsymbol{u}_n - 2T\boldsymbol{v}_{nk} \\
\boldsymbol{t}_{PREP,k} &\leq \boldsymbol{t}_{PREP,n} - \Delta t_{PREP} - \ 2T\boldsymbol{u}_n + \ T\boldsymbol{r}_n \ + 2T \\
\boldsymbol{t}_{PREP,k} &\geq \boldsymbol{t}_{PREP,n} + \Delta t_{PREP} + \ 2T\boldsymbol{u}_n - \ T\boldsymbol{s}_n \ - 2T \\
&\forall n \in \mathcal{N}, \ \forall k \in \mathcal{N}; \ k > n
\end{aligned} \tag{4.5.28}$$

4.5.7 Preparation Scheduling

The constraints in equation 4.5.28 are only valid for the case where buffers n and k are prepared in the same slot. Recall that we defined a variable, $\boldsymbol{a}_{nk}$, which indicated if this is the case. Accordingly, the final preparation scheduling constraint is a modification of equation 4.5.28 which uses the *big-M* method, with $\mathbb{M} = 2T$, to disable all of the constraints when $\boldsymbol{a}_{nk} = 0$. The final preparation constraint is given in equation equation 4.5.29 below.

$$\begin{aligned}
\boldsymbol{t}_{PREP,k} &\leq \boldsymbol{t}_{PREP,n} - \Delta t_{PREP} - \ 2T\boldsymbol{a}_{nk} + 2T\boldsymbol{u}_n - 2T\boldsymbol{v}_{nk} + 4T \\
\boldsymbol{t}_{PREP,k} &\geq \boldsymbol{t}_{PREP,n} + \Delta t_{PREP} + \ 2T\boldsymbol{a}_{nk} - 2T\boldsymbol{u}_n - 2T\boldsymbol{v}_{nk} - 2T \\
\boldsymbol{t}_{PREP,k} &\leq \boldsymbol{t}_{PREP,n} - \Delta t_{PREP} - \ 2T\boldsymbol{a}_{nk} - 2T\boldsymbol{u}_n + \ T\boldsymbol{r}_n \ + 4T \\
\boldsymbol{t}_{PREP,k} &\geq \boldsymbol{t}_{PREP,n} + \Delta t_{PREP} + \ 2T\boldsymbol{a}_{nk} + 2T\boldsymbol{u}_n - \ T\boldsymbol{s}_n \ - 4T \\
&\forall n \in \mathcal{N}, \ \forall k \in \mathcal{N}; \ k > n
\end{aligned} \tag{4.5.29}$$

A truth table for equation 4.5.29 is given in Table 4.8, with '$*$' denoting the binary set, $\{0, 1\}$. Recall also that equation 4.5.11 already ensures $0 \leq \boldsymbol{t}_{PREP,k} \leq T$, so we do not need duplication of that constraint in equation 4.5.29.

Table 4.8: Truth table for equation 4.5.29

$\boldsymbol{a}_{nk}$	$\boldsymbol{u}_n$	$\boldsymbol{v}_n$	$\boldsymbol{r}_n$	$\boldsymbol{s}_n$	$\boldsymbol{t}_{PREP,k} = \bullet$
0	$*$	$*$	$*$	$*$	(no active scheduling constraints)
1	0	0	$*$	$*$	$\boldsymbol{t}_{PREP,n} + \Delta t_{PREP} \leq \bullet$
1	0	1	$*$	$*$	$\bullet \leq \boldsymbol{t}_{PREP,n} - \Delta t_{PREP}$
1	1	$*$	0	0	$\boldsymbol{t}_{PREP,n} + \Delta t_{PREP} \leq \bullet \leq \boldsymbol{t}_{PREP,n} - \Delta t_{PREP}$
1	1	$*$	0	1	$\boldsymbol{t}_{PREP,n} + \Delta t_{PREP} - T \leq \bullet \leq \boldsymbol{t}_{PREP,n} - \Delta t_{PREP}$
1	1	$*$	1	0	$\boldsymbol{t}_{PREP,n} + \Delta t_{PREP} \leq \bullet \leq \boldsymbol{t}_{PREP,n} - \Delta t_{PREP} + T$
1	1	$*$	1	1	$\boldsymbol{t}_{PREP,n} + \Delta t_{PREP} - T \leq \bullet \leq \boldsymbol{t}_{PREP,n} - \Delta t_{PREP} + T$

4.5.8 Complete Model Summary

The complete model is summarised below. The constraint equations have been rearranged so that variables are on the left hand side and constants are on the right.

Note that variables $\boldsymbol{a}_{nk}$, $\boldsymbol{t}_{PREP,n}$, $\boldsymbol{t}_{PREP,k}$, $\boldsymbol{t}_{LOWER,n}$ and $\boldsymbol{t}_{UPPER,n}$ are not required in the summary model. Values for these variables can be obtained, post solution, by applying equations 4.5.7, 4.5.10, 4.5.10 (replacing n with k), 4.5.17 and 4.5.20 respectively.

Minimise:

$$\boldsymbol{Z} = \sum_{m \in \mathcal{M}} \sum_{p \in \mathcal{P}} c_m \boldsymbol{y}_{mp} \tag{4.3.3}$$

Subject to:

$$\sum_{p \in \mathcal{P}} \boldsymbol{x}_{np} = 1 \quad \forall n \in \mathcal{N} \tag{4.4.3}$$

$$\sum_{m \in \mathcal{M}} \boldsymbol{y}_{mp} \leq 1 \quad \forall p \in \mathcal{P} \tag{4.4.4}$$

$$U_n \boldsymbol{x}_{np} - \sum_{m \in \mathcal{M}} V_m \boldsymbol{y}_{mp} \leq 0 \quad \forall n \in \mathcal{N},\ \forall p \in \mathcal{P} \tag{4.4.5}$$

$$V_{MAX} \boldsymbol{x}_{np} + f_{MINFILL} \sum_{m \in \mathcal{M}} V_m \boldsymbol{y}_{mp} \leq U_n + V_{MAX} \quad \forall n \in \mathcal{N},\ \forall p \in \mathcal{P} \tag{4.4.8}$$

$$\Delta t_{PREP} \sum_{n \in \mathcal{N}} \boldsymbol{x}_{np} \leq f_{UTIL} T \quad \forall p \in \mathcal{P} \tag{4.4.10}$$

$$\begin{aligned} \boldsymbol{z}_n \leq T - \Delta t_{HOLD,PRE} - \Delta t_{TRANSFER} - \Delta t_{USE,n} - \Delta t_{HOLD,POST} \\ \forall n \in \mathcal{N} \end{aligned} \tag{4.5.2}$$

$$\begin{aligned} 2\boldsymbol{w}_{nkp} - \boldsymbol{x}_{np} - \boldsymbol{x}_{kp} &\leq \ -2 \\ \boldsymbol{w}_{nkp} - \boldsymbol{x}_{np} - \boldsymbol{x}_{kp} &\geq \ -1 \end{aligned} \quad \forall n \in \mathcal{N}, \ \ \forall k \in \mathcal{N}; \ k > n, \ \ \forall p \in \mathcal{P} \tag{4.5.5}$$

$$\begin{aligned} T\boldsymbol{q}_n - \boldsymbol{z}_n &\geq -t_{USE,n} \\ T\boldsymbol{q}_n - \boldsymbol{z}_n &\leq -t_{USE,n} + T \end{aligned} \quad \forall n \in \mathcal{N} \tag{4.5.13}$$

$$\begin{aligned} -T\boldsymbol{q}_n + T\boldsymbol{r}_n + \boldsymbol{z}_n &\leq t_{USE,n} + \Delta t_{PREP} \\ -T\boldsymbol{q}_n + T\boldsymbol{r}_n + \boldsymbol{z}_n &\geq t_{USE,n} + \Delta t_{PREP} - T \end{aligned} \quad \forall n \in \mathcal{N} \tag{4.5.30}$$

$$\begin{aligned} T\boldsymbol{q}_n + T\boldsymbol{s}_n - \boldsymbol{z}_n &\leq -t_{USE,n} + \Delta t_{PREP} \\ T\boldsymbol{q}_n + T\boldsymbol{s}_n - \boldsymbol{z}_n &\geq -t_{USE,n} + \Delta t_{PREP} + T \end{aligned} \quad \forall n \in \mathcal{N} \tag{4.5.31}$$

$$\begin{aligned} \boldsymbol{r}_n + \boldsymbol{s}_n - \ \ \boldsymbol{u}_n &\geq 0 \\ \boldsymbol{r}_n + \boldsymbol{s}_n - 2\boldsymbol{u}_n &\leq 0 \end{aligned} \quad \forall n \in \mathcal{N} \tag{4.5.27}$$

$$\begin{aligned}
T\boldsymbol{q}_k - T\boldsymbol{q}_n - 2T\boldsymbol{u}_n + 2T\boldsymbol{v}_{nk} + 2T\sum_{p\in\mathcal{P}}\boldsymbol{w}_{nkp} - \boldsymbol{z}_k + \boldsymbol{z}_n& \\
\leq t_{USE,n} - t_{USE,k} - \Delta t_{PREP} + 4T& \\
T\boldsymbol{q}_k - T\boldsymbol{q}_n + 2T\boldsymbol{u}_n + 2T\boldsymbol{v}_{nk} - 2T\sum_{p\in\mathcal{P}}\boldsymbol{w}_{nkp} - \boldsymbol{z}_k + \boldsymbol{z}_n& \\
\geq t_{USE,n} - t_{USE,k} + \Delta t_{PREP} - 2T& \\
T\boldsymbol{q}_k - T\boldsymbol{q}_n - T\boldsymbol{r}_n + 2T\boldsymbol{u}_n + 2T\sum_{p\in\mathcal{P}}\boldsymbol{w}_{nkp} - \boldsymbol{z}_k + \boldsymbol{z}_n& \qquad (4.5.32) \\
\leq t_{USE,n} - t_{USE,k} - \Delta t_{PREP} + 4T& \\
T\boldsymbol{q}_k - T\boldsymbol{q}_n + T\boldsymbol{s}_n - 2T\boldsymbol{u}_n - 2T\sum_{p\in\mathcal{P}}\boldsymbol{w}_{nkp} - \boldsymbol{z}_k + \boldsymbol{z}_n& \\
\geq t_{USE,n} - t_{USE,k} + \Delta t_{PREP} - 4T& \\
\forall n\in\mathcal{N},\ \forall k\in\mathcal{N};\ k>n&
\end{aligned}$$

Where:

$$\boldsymbol{q}_n \in \{0,1\} \quad \forall n\in\mathcal{N} \qquad (4.5.12)$$

$$\boldsymbol{r}_n \in \{0,1\} \quad \forall n\in\mathcal{N} \qquad (4.5.19)$$

$$\boldsymbol{s}_n \in \{0,1\} \quad \forall n\in\mathcal{N} \qquad (4.5.22)$$

$$\boldsymbol{u}_n \in \{0,1\} \quad \forall n\in\mathcal{N} \qquad (4.5.26)$$

$$\boldsymbol{v}_{nk} \in \{0,1\} \quad \forall n\in\mathcal{N},\ \forall k\in\mathcal{N};\ k>n \qquad (4.5.15)$$

$$\boldsymbol{w}_{nkp} \in \{0,1\} \quad \forall n\in\mathcal{N},\ \forall k\in\mathcal{N};\ k>n,\ \forall p\in\mathcal{P} \qquad (4.5.4)$$

$$\boldsymbol{x}_{np} \in \{0,1\} \quad \forall n\in\mathcal{N},\ \forall p\in\mathcal{P} \qquad (4.4.1)$$

$$\boldsymbol{y}_{mp} \in \{0,1\} \quad \forall m\in\mathcal{M},\ \forall p\in\mathcal{P} \qquad (4.3.1)$$

$$\Delta t_{HOLD,MIN} \leq \boldsymbol{z}_n \leq \Delta t_{HOLD,MAX}; \quad \boldsymbol{z}_n \in \mathbb{R} \quad \forall n \in \mathcal{N} \tag{4.5.1}$$

4.6 Further Optimisation

4.6.1 Goal programming

The complete model summarised in Section 4.5.8 may be solved to give the required number of vessels that correspond to the minimum cost. As was alluded to in Section 5.1, there may be many possible feasible solutions that correspond to this minimum cost and would yield the same set of vessels (though the selected preparation vessels may not necessarily be used to prepare the same buffers). Thus, there may be many ways to represent an optimal process schedule.

If, for example, the data described in Chapter 3 were used as an input to the complete model, an *equipment time utilisation* plot (see Section 5.1) for the feasible optimal schedule (such as that shown in Figure 4.3) may be generated.

Equipment time utilisation plots are discussed in more detail in Chapter 5; they show the scheduling of procedures or operations (bars) in equipment (broken lines), coloured by buffers. They show a single-cycle window at steady-state.

Recall that one of our decision variables is $\boldsymbol{z}_n$, the buffer hold duration. In Figure 4.3, we can see that Buffer #2 is scheduled such that its hold procedure is bottlenecked, i.e. the start of the hold procedure for a given batch coincides precisely with the end of the hold procedure for the previous batch. For this buffer, it can also be seen that there exists some free time after its preparation procedure.

Delaying the preparation of Buffer #2 by e.g. three hours would also give an optimal solution, with the vessel selection unchanged, and would prevent the hold procedure for Buffer #2 from being bottlenecked. There exists sufficient free time in the 8000 l vessel to do this. Note that delaying a preparation by some amount δ is equivalent to reducing $\boldsymbol{z}_n$ by δ. Recall also that $\boldsymbol{z}_n$ has a lower bound, $\Delta t_{HOLD,MIN}$.

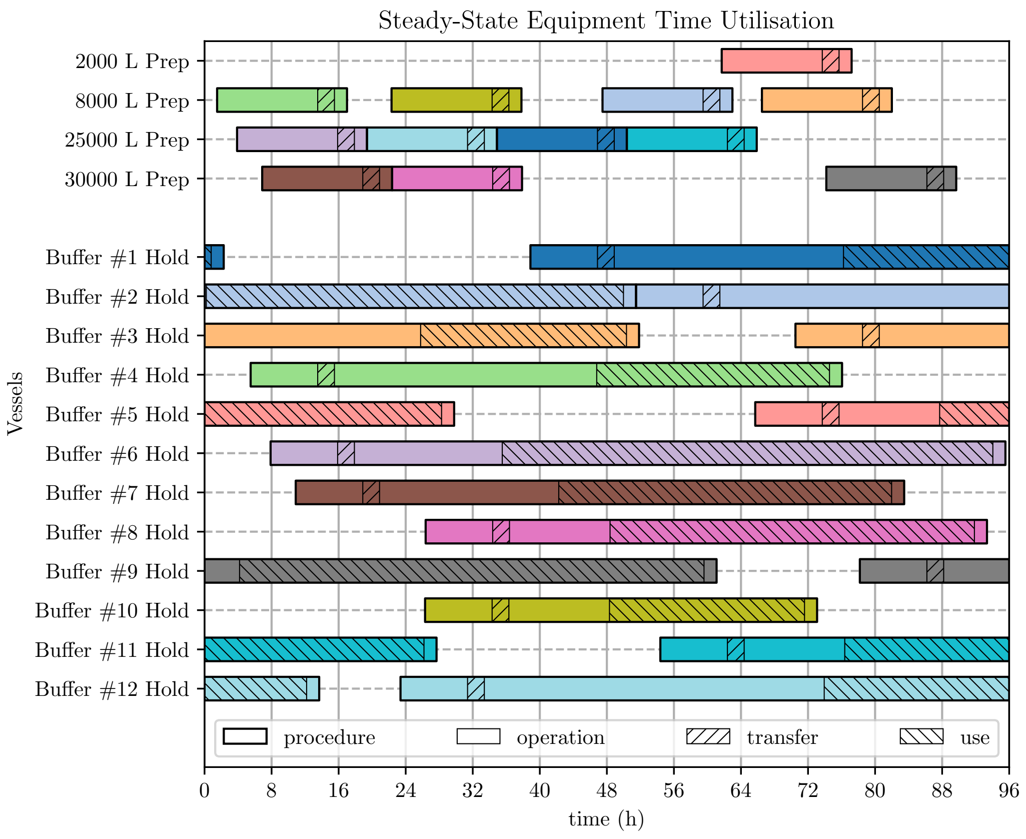


Figure 4.3: Random example – primary objective (note that buffers are individually coloured; each buffer has a dedicated hold vessel, from which its colour may be determined)

In terms of visualising a realistic schedule, it is desirable to minimise the hold durations, i.e. we wish to define a new objective function, to be minimised:

$$\boldsymbol{Y} = \sum_{n \in \mathcal{N}} \boldsymbol{z}_n \tag{4.6.1}$$

Note that we want to maintain the original optimum objective function value, i.e. we define:

$$Z' = \min \boldsymbol{Z} \tag{4.6.2}$$

In equation 4.6.2, Z' is the optimal objective function value obtained by solving the complete model. Note that Z' is not in bold, as it is treated as a constant parameter in our second-pass model. The complete model is then re-run with the original objective function replaced by the new objective, equation 4.6.1, with the following constraint added:

$$\sum_{m \in \mathcal{M}} \sum_{p \in \mathcal{P}} c_m \boldsymbol{y}_{mp} = Z' \tag{4.6.3}$$

Implementing the secondary goal programming model on the same data-set yields the plot shown in Figure 4.4. Note that the buffer hold procedure for Buffer #2 is no longer bottlenecked. We can also see that the hold times for Buffers #1, #4, #6, #7 and #12 have all decreased, while hold times for the remaining buffers are unchanged. Overall, there has been a reduction in total hold time, which was our aim. Note also that, although the same preparation vessels are selected (i.e. the same minimal cost is obtained), some buffers are now prepared in different vessels to those originally assigned. For example, in the original result, Buffers #1 and #6 were prepared in the 25 000 l vessel and with the secondary goal added, they are now prepared in the 30 000 l vessel. Conversely, preparation of Buffer #7 shifts from the 30 000 l vessel to the 25 000 l vessel.

4.6.2 Secondary Model Summary

The secondary model is summarised below:

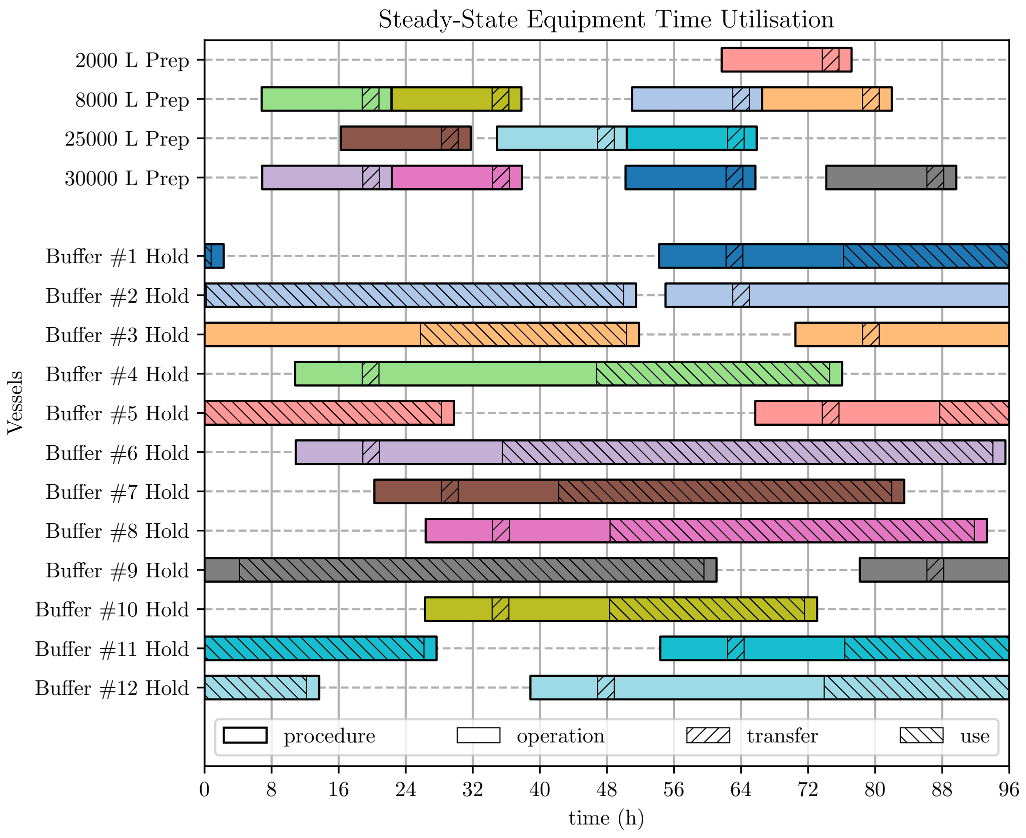


Figure 4.4: Random example – secondary objective

Minimise:

$$\boldsymbol{Y} = \sum_{n \in \mathcal{N}} \boldsymbol{z}_n \tag{4.6.1}$$

Subject to all constraints summarised in Section 4.5.8, and additionally:

$$\sum_{m \in \mathcal{M}} \sum_{p \in \mathcal{P}} c_m \boldsymbol{y}_{mp} = Z' \tag{4.6.3}$$

Where:

$$\boldsymbol{q}_n \in \{0,1\} \quad \forall n \in \mathcal{N} \tag{4.5.12}$$

$$\boldsymbol{r}_n \in \{0,1\} \quad \forall n \in \mathcal{N} \tag{4.5.19}$$

$$\boldsymbol{s}_n \in \{0,1\} \quad \forall n \in \mathcal{N} \tag{4.5.22}$$

$$\boldsymbol{u}_n \in \{0,1\} \quad \forall n \in \mathcal{N} \tag{4.5.26}$$

$$\boldsymbol{v}_{nk} \in \{0,1\} \quad \forall n \in \mathcal{N},\ \forall k \in \mathcal{N};\ k > n \tag{4.5.15}$$

$$\boldsymbol{w}_{nkp} \in \{0,1\} \quad \forall n \in \mathcal{N},\ \forall k \in \mathcal{N};\ k > n,\ \forall p \in \mathcal{P} \tag{4.5.4}$$

$$\boldsymbol{x}_{np} \in \{0,1\} \quad \forall n \in \mathcal{N},\ \forall p \in \mathcal{P} \tag{4.4.1}$$

$$\boldsymbol{y}_{mp} \in \{0,1\} \quad \forall m \in \mathcal{M},\ \forall p \in \mathcal{P} \tag{4.3.1}$$

$$\Delta t_{HOLD,MIN} \leq \boldsymbol{z}_n \leq \Delta t_{HOLD,MAX}; \quad \boldsymbol{z}_n \in \mathbb{R} \quad \forall n \in \mathcal{N} \tag{4.5.1}$$

4.7 Implementation

Thus far, this chapter has dealt with the mathematics underpinning the model. This section describes the code which was used to obtain solutions to the equations derived above.

The python programming language was used for all model and plotting code. There were several reasons for this choice. For speed of development, an interpreted language is preferred. Since the bulk of the computation is performed by solver packages, the slower speed of an interpreted language is not an important consideration. There exist several useful open-source plotting and statistical libraries for python, allowing both generation and manipulation of the results.

The git[1] version control software package was used to manage the development, including the LaTeX scripts that comprise this document. The git repository for this project is hosted on GitHub, at `https://github.com/multipitch/dissertation`

Input data files are either in `csv` format (for tabular data) or `ini` format (for parameters), as parsers for both of these formats are included in the standard python library.

4.7.1 MILP Solvers

There are several MILP solver packages available that can be used to solve problems such as the basic, complete and secondary problems summarised in this chapter. Both proprietary and open-source MILP solvers exist. Proprietary solvers include Gurobi[2], CPLEX[3] and XPRESS[4]. Open source solvers include GLPK[5] and Cbc[6].

Initial coding efforts centred around CPLEX, since it has a free and readily available academic license, good documentation and a well-documented python API.

One area of interest is the comparison of different MILP solvers, so a more general modelling approach was required. It was also noted during development

[1] *git*, `https://git-scm.com/`

[2] *Gurobi Optimizer*, `http://www.gurobi.com/products/gurobi-optimizer`

[3] *IBM ILOG CPLEX Optimizer* , `https://www.ibm.com/software/commerce/optimization/cplex-optimizer/`

[4] *FICO Xpress Solver*, `http://www.fico.com/en/products/fico-xpress-solver`

[5] *GLPK*, `https://www.gnu.org/software/glpk/`

[6] *Cbc*, `https://projects.coin-or.org/Cbc`

that the bulk of the code dealt with the construction of the model (reading from input files and constructing the equations summarised in Sections 4.5.8 and 4.6.2); less than ten lines of code involved interfacing with the solver. Accordingly, development was switched away from the CPLEX API and towards the PuLP[7] API. PuLP is an open-source python library that provides an API to many proprietary and open-source MILP solvers. This change provided a pathway for comparing the effectiveness of several different solvers.

Ultimately, the solvers investigated were CPLEX (version 12.7.1), Cbc (as bundled with version 1.6.7 of PuLP) and GLPK (version 4.63).

4.7.2 Development Environment

Development was mainly carried on a computer running Arch Linux 64 bit. This is a rolling-release operating system and so does not have a version number.

The current (at time of writing) available version of CPLEX is version 12.7.1. This version is not compatible with current python versions (3.6 and above). To overcome this, the pyenv[8] package was used to locally install python version 3.5.1 for use with the working repository. Using a local pyenv version in this manner required the installation of local static versions of all libraries used in the code which is advantageous in preventing breakages during software updates (these can be common with rolling-release operating systems).

The initial version of PuLP used during development was version 1.6.6. This version was found to be incompatible with CPLEX version 12.7.1 due to a change in the CPLEX python API between versions 12.7.0 and 12.7.1. A patch to PuLP was required to maintain inter-operability. As development progressed, version 1.6.7 of PuLP was released which contained the same patch, restoring CPLEX compatibility. At this point, version 1.6.7 of PuLP was installed unmodified. All other software was also installed un-patched and unmodified. Table 4.9 lists the relevant third-party python libraries installed.

[7] *PuLP*, `https://pypi.python.org/pypi/PuLP`

[8] *pyenv* `https://github.com/pyenv/pyenv`

[9] *matplotlib*, `https://matplotlib.org/`

Table 4.9: Installed third-party python libraries

library	version	reason for installation
matplotlib[9]	2.0.2	generation of plots
numpy[10]	1.13.0	working with arrays
pulp	1.6.7	API for multiple MILP solvers
pylatexenc[11]	1.2	converting strings to a LaTeX-compatible format
scipy[12]	0.19.1	calculating binomial coefficients for complexity plots

Using Linux, the source code repository may be checked out from GitHub by installing git[13] and running the following command in a terminal:

```
$ git clone https://github.com/multipitch/dissertation
```

The repository contains three folders; `examples`, `src`, and `tex`. All code is contained in the `src` folder. The `examples` folder contains some example input files and the `tex` folder contains the LaTeX scripts and associated files that were used to generate this document.

The `src` folder contains three files:

- `model.py` contains the code specifying and solving models
- `plots.py` contains code for generating various plots
- `runmodel` is a short executable script that acts as a command-line front-end for running the models in `model.py`

A brief description of the code in `model.py` and `plots.py` is given in as an appendix to this report.

[10]*numpy* `http://www.numpy.org/`
[11]*pylatexenc* `https://pypi.python.org/pypi/pylatexenc`
[12]*scipy* `https://www.scipy.org/`
[13]*git* `https://git-scm.com/`

Chapter 5

Results

Bosh! Stephen said rudely. A man of genius makes no mistakes.
His errors are volitional and are the portals of discovery.

— James Joyce, *Ulysses*

5.1 Output Data

The primary aim is to generate a list of the required preparation volumes. For the random example used in this chapter, solution of the model yields the results in Table 5.1.

Table 5.1: Required preparation vessels for random example

vessel size
2000 l
8000 l
25 000 l
30 000 l

While Table 5.1 gives the essential information allowing the buffer preparation area to be sized and costed, it may be more instructive to return a matrix showing where each buffer is to be prepared. Indeed, the presentation of

Table 5.2: Buffer / vessel matrix for random example

	2000 l	8000 l	25 000 l	30 000 l
Buffer #1			●	
Buffer #2		●		
Buffer #3		●		
Buffer #4		●		
Buffer #5	●			
Buffer #6			●	
Buffer #7				●
Buffer #8				●
Buffer #9				●
Buffer #10		●		
Buffer #11			●	
Buffer #12			●	

such a minimal manifestation of results is unlikely to convince a client of the feasibility of the solution.

Table 5.2 shows *one possible* vessel assignment for the random example. Note that the existence of more than one feasible version of Table 5.1 is unlikely for a given data-set, especially if vessel cost scales non-linearly with vessel volumes. On the other hand, many feasible versions of Table 5.2 may exist; e.g. it may be possible to switch the vessels in which buffers are prepared and still end up with the same optimal vessel selection. This phenomenon is dealt with in more detail in Section 4.6.

While Table 5.2 does give more information than Table 5.1, it still doesn't visibly confirm to the reader that the solution is feasible. An *equipment time utilisation* plot provides a clear visual illustration of a solution and displays a feasible schedule. The explanatory plot in Figure 3.1 is an example of an equipment time utilisation plot.

An equipment time utilisation plot for the random example data detailed in Chapter 3 is shown in Figure 5.1. Each horizontal dotted line represents a

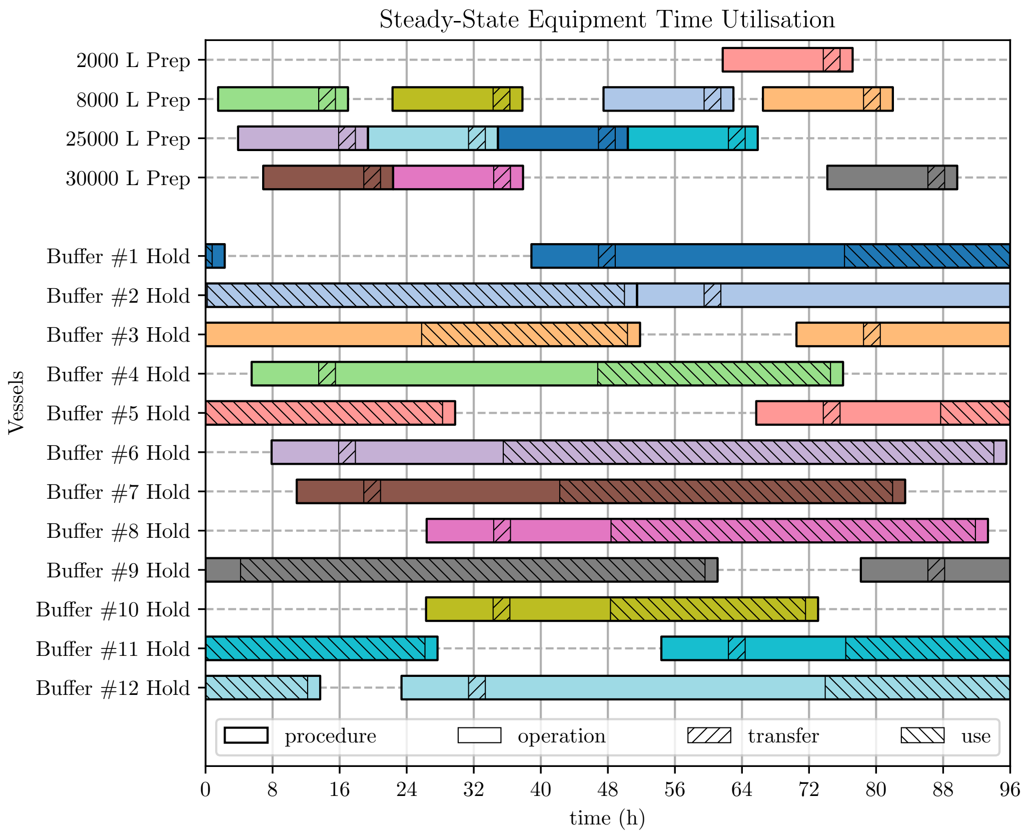


Figure 5.1: Equipment time utilisation for random example

piece of equipment (preparation or hold vessels in this case). The presence of a bar on a dotted line indicates that the respective piece of equipment is in use. The hatched bars indicate transfers, as per the legend.

Transfer from a buffer hold vessel to the production process is shown as a contiguous bar in the hold vessel, representing the period from the start of the first use to the end of the last use of the buffer in a given batch, although the demand from the production process may be discontinuous.

The bars are coloured by buffer. Since the buffer hold vessels are dedicated to particular buffers, and are named according to the buffers they contain, the buffer hold entries serve as a legend for the colour scheme.

Note that the time window shown in Figure 5.1 displays a single cycle at steady-state; the next cycle and the previous cycle would be identical. The offset of the single-cycle window is somewhat arbitrary; the visible window can be thought of as a cylinder that has been cut at right a angles to its ends and flattened out.

It should be noted also that Figure 5.1 does not highlight the batches to which the procedures belong. The entire downstream process may take more than one cycle to complete, so the window visible in the plot could be showing buffer preparation and hold procedures from several successive batches.

The visual output of an equipment time utilisation plot provides a useful method for validating results. As the model detailed in Chapter 4 was being developed, such plots were a useful way of highlighting logical flaws or bugs in the code. With the finished model, such plots are a useful tool for convincing colleagues and clients that a proposed vessel selection is indeed feasible.

5.2 Solution Duration

5.2.1 Variation of Problem Size

Using the parameters data set given in Table 3.3 and the vessels data set given in Table 3.1, the time taken for the CPLEX solver to reach a feasible solution

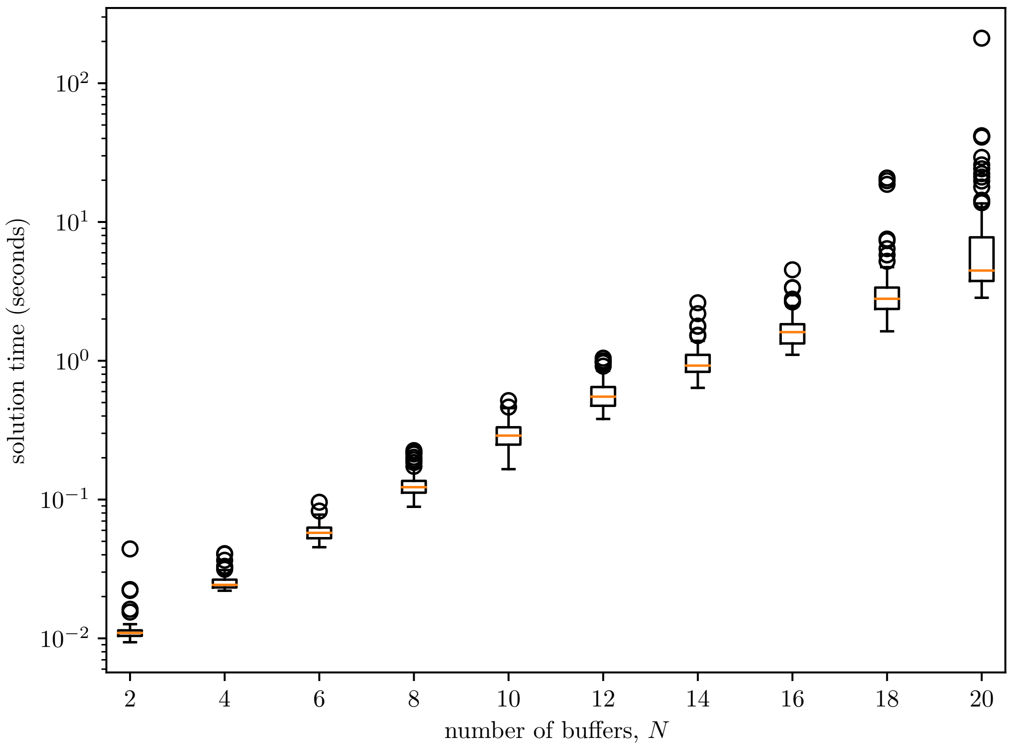


Figure 5.2: Box plots showing solution time as a function of buffer count, using CPLEX. 100 randomly-generated runs were performed for each value of N.

was evaluated for a range of randomly-generated buffer data sets of different sizes.

The runs were carried out on a computer with 16 GB of RAM and an Intel 3770K processor (four physical cores; eight virtual cores, running at 3.5 GHz to 3.9 GHz). The operating system was Arch Linux 64 bit (rolling release). It was note that the CPLEX solver uses multi-threading effectively, as all eight virtual cores were running at 100 % fairly consistently during the runs (the usage tended to dip slightly at the start and end of each run). RAM usage during the runs was below 1 GB, so memory shortage was not a factor in the solution durations.

For a set of problem sizes (i.e. values of N), 100 random data sets were generated for each problem size and the solution durations were recorded. A box plot of the durations is shown in Figure 5.2.1. A total of 1000 runs were performed to generate Figure 5.2.1 and for each run, an optimal solution was found. Note that there are several slow outliers for each problem size. In the case of $N = 20$, the highest outlier is approximately two orders of magnitude greater than the mean duration, indicating a high degree of variability.

Such variability in solution duration is not uncommon in MILP problems. Methods for solving problems with binary variables necessarily involve some searching through large binary trees, where search time can vary greatly depending on the location of the optimum result and the search strategy.

5.2.2 Comparison of Solvers

A similar analysis to that carried out in Section 5.2.1 was attempted with the Cbc and GLPK solvers. The results are shown in Figure 5.2.2. Each point represents the mean duration of 100 runs of a particular problem size with a particular solver.

Both Cbc and GLPK were found to be considerably slower than CPLEX.

The GLPK solver is only capable of running single-threaded so cannot take advantage of a multi-threaded processor. It was the fastest at solving the trivial two-buffer problem, but showed the steepest rate of solution time increase with increasing number of buffers. At a size of $N = 8$, GLPK showed massive variation in solution time, with solutions ranging from less than 10 seconds to over 8 hours. As a result, only data points for $N = 6$ and below are shown for GLPK.

The Cbc solver had better performance but had reliability issues; on three random runs out of 100, with a problem size of $N = 10$, Cbc was taking several orders of magnitude longer than the mean to obtain a solution, so those runs were halted. While the CPLEX and GLPK solvers were called with no arguments, the `threads=8` argument was passed to the Cbc solver to get it to use all virtual cores on the computer (CPLEX uses all virtual cores by default and GLPK only operates as a single thread).

CPLEX proved robust over the range $2 \leq N \leq 20$.

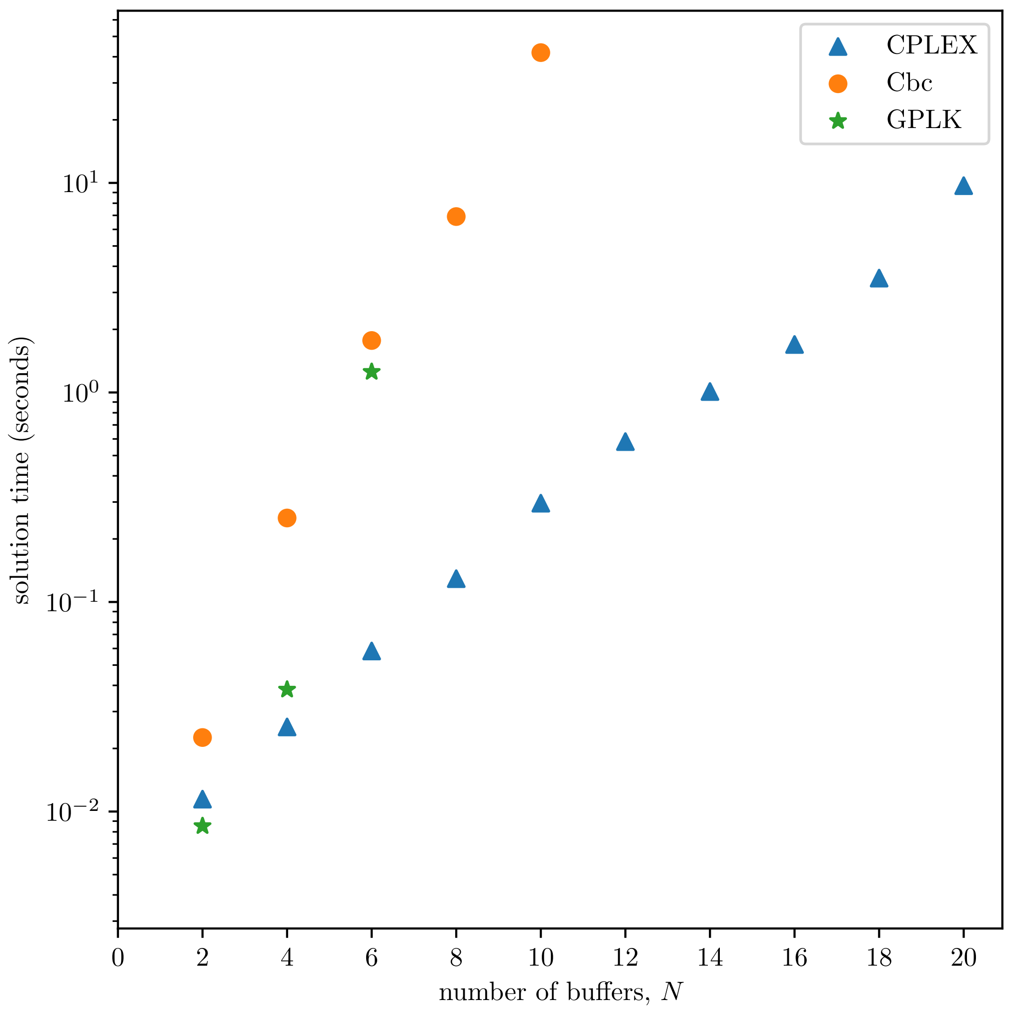


Figure 5.3: Solution time as a function of buffer count – each data point represents the mean solution time of 100 runs of size N using the solvers indicated in the legend.

Note that the data in Figure 5.2.2 indicate that solution time is exponential in N, which is to be expected for an **NP**-hard optimisation problem.

5.2.3 Real-World Data

The model was applied to (obfuscated) data from two real-world data sources. In both cases, CPLEX found feasible solutions in under one minute. The data and results for both real-world examples are recorded in an appendix to this report.

Chapter 6

Discussion

Nothing in life is as important as you think it is when you are thinking about it.

— Daniel Kahneman, *Thinking, Fast and Slow*

6.1 Comment on Results

The results obtained show that the model is suitable for generating optimal buffer preparation designs. For problem sizes in the expected range of 10–20 buffers, it is important to note that an exact solution can be obtained in a reasonable amount of time. This is a significant result; prior to carrying out this exercise, it was not known if an optimum solution could feasibly be obtained.

The use of the proprietary CPLEX solver produced the fastest solutions and produced a feasible solution for all problems examined. The Cbc solver was less robust; on several occasions it crashed while running. The worst performance was obtained from the GLPK solver, which appears incapable of arriving at an optimum solution in under an hour for a medium sized problem.

Table 6.1: Model complexity

model	no. of variables	no. of equations
basic	$N^2 + NM$	$2N^2 + 3N + 1$
complete	$N\binom{N}{2} + N^2 + \binom{N}{2} + NM + 5N$	$2N\binom{N}{2} + 4\binom{N}{2} + 2N^2 + 12N + 1$

Table 6.2: Simplified model complexity

model	no. of variables	no. of equations
basic	$2N^2$	$2N^2 + 3N + 1$
complete	$\frac{1}{2}N^3 + \frac{5}{2}N^2 + 5N$	$N^3 + 4N^2 + 12N + 1$

6.2 Challenges

The most challenging part of this study was the development of the constraint equations required for the complete model, particularly in the area of cyclic task scheduling. The PuLP API is easy to use and well documented, so building the model in python was a more straightforward task.

6.3 Complexity

The notation for complexity in this section is according to Knuth (1976).

The problem complexity depends on both the number of buffers, N, and the number of vessels, M. The complexity may be evaluated in terms of both the number of variables and the number of equations in the problem. The complexities of the basic and complete problems are summarised in Table 6.1. For example, a complete problem with twelve buffers and twelve vessels has 2281 equations in 1206 variables.

Noting that there is a weak dependence on M and that, as the problem grows larger in terms of N, it is likely that M will reach a maximum value, the approximation $M \approx N$ may be used as a conservative simplification when considering complexity. This simplification has been used to generate the plots in Figures 6.1 and 6.2. The complexity can be further simplified by noting that for the binomial coefficient term,

$$\lim_{N \to \infty} \binom{N}{2} = \tfrac{1}{2} N^2 \tag{6.3.1}$$

Applying both approximations gives the simplified complexities tabulated in Table 6.2.

For the basic model, we can see that both the number of variables and the number of equations is $O\left(N^2\right)$.

For the complete model, the number of variables and the number of equations are both $O\left(N^3\right)$.

The addition of the scheduling constraints in the complete model requires the comparison of pairs of buffers, which is $O\left(N^2\right)$. These comparisons occur for each slot, giving a multiplicative factor of $O\left(N\right)$, leading to an overall complexity of $O\left(N^3\right)$.

As was mentioned in Section 5.2.2, the time complexity of the complete problem is exponential in N, which indicated the complete model is (NP)-hard. It was noted in Section 4.1 that the problem appeared to bear similarities to the bin-packing problem, which is also known to be **NP**-hard (Korte and Vygen, 2012), so this result was as expected.

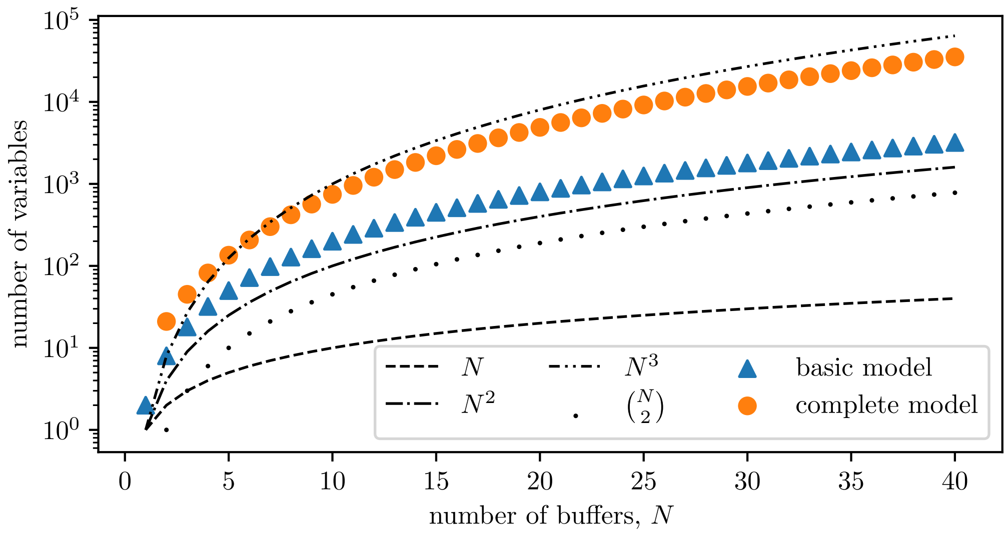


Figure 6.1: Model complexity – variables

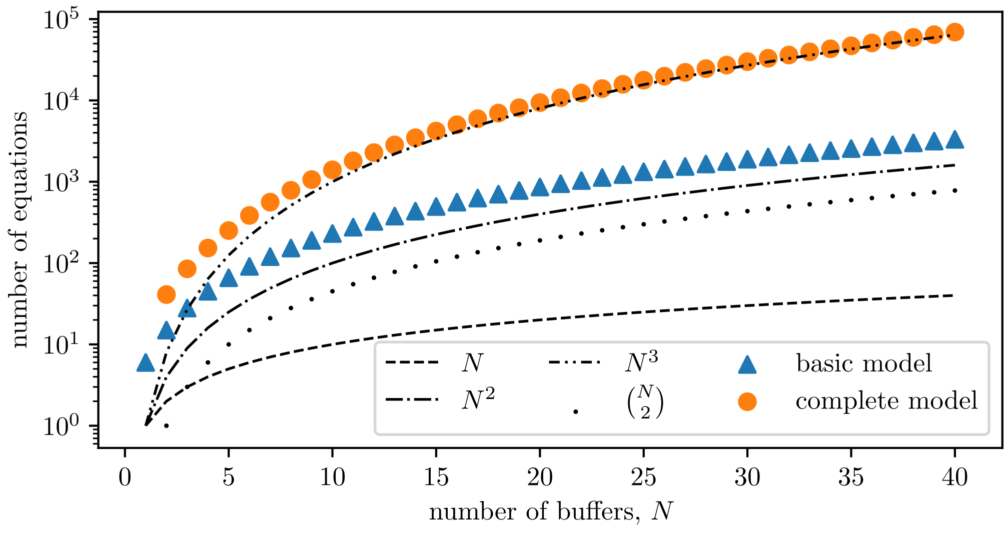


Figure 6.2: Model complexity – equations

Chapter 7

Conclusions

I was not proud of what I had learned but I never doubted that it was worth knowing.

— Hunter S. Thompson, *The Rum Diary*

The models derived in Chapter 4 succeed in describing the buffer vessel selection problem. The code which implements the models produces feasible solutions from input data sets and gives both quantitative outputs (values for all variables) and useful visual output (the equipment time utilisation plot).

Out of the MILP solvers investigated, it was found that CPLEX provided the fastest and most robust solutions.

The number of equations in the complete model was found to be cubic with respect to the number of buffers. The number of variables in the complete model was also found to be cubic in the number of buffers. The time complexity of the complete problem was found to be exponential in the number of vessels.

The basic model represents the simplest manifestation of the buffer vessel selection problem, requiring a minimal amount of information Additionally, the basic model has been built upon to arrive at the complete model, which accounts for scheduling and accordingly requires some input data on timings.

The secondary model builds upon the complete model and uses goal programming to provide a more realistic schedule. Note that it does not improve on the primary objective function value, i.e. the minimisation of vessel costs.

The models contain a number of simplifications. Some of these simplifications are required due to the probable lack of sufficient information. One such example is fixed operation durations for operations such as transfers. If only a single model of pump were available, a flow rate could be defined, so the transfer duration would scale linearly with the buffer volume. In reality, given the range of volumes in a typical process, several sizes of pumps and pipework may be specified so that pumps aren't oversized for small transfers and pumps aren't so small that large transfers take an unacceptable length of time. As a result, the assumption that transfer duration is fixed allows the selection of a conservative transfer time to cover all eventualities.

Another assumption is that buffer volumes are fixed. For example, if a buffer is used in two operations which occur in different procedures at very different times in the cycle, we do not allow the possibility that we may split this buffer into two separate preparations. If, for instance, the particular buffer was the largest-volume buffer being prepared by some margin, it may make sense to split it into two batches to reduce the maximum size of preparation vessel required. There are trade-offs here between the saving from reducing the volume of the largest reparation vessel and the increased operating costs from performing an extra preparation, in terms of personnel and the cost of cleaning utilities. Apart from the difficulty for building such flexibility into the model, the objective function would potentially become a good deal more difficult to define, requiring knowledge on trade-offs between operating versus capital cost. Such knowledge may not be readily available.

One additional feature which warrants further work is material compatibility. Certain buffers may require the use of high-nickel alloys instead of the usual grades of stainless steel; these alloys may be several times the price of stainless steel. A more resistant vessel can be used to prepare all buffers that may be prepared in a less resistant vessel. This would require a compatibility matrix between buffers and vessel materials (the latter being a new dimension). Additional logic would also be required to prevent incorrect vessel material selection.

Other scope for further work includes the application of the model to multiproduct facilities. Generally, a multi-product facility will run a campaign of batches of one product, then switch over to run a campaign of some other process. Such a facility would need buffer vessels capable of supporting each process.

Further exploration of results is possible through e.g. Monte Carlo simulation. For instance, the sensitivity of a solution to variation in input data could be explored, which would give a client greater certainty that a proposed solution would be able to handle expected variations.

The methodology detailed in this report will be of use in future bioprocessing facility design projects and adds to the body of knowledge in the fields of process engineering and linear programming.

Program installation and usage

- Python should be installed, along with the third-party python libraries listed in Table 4.9. Note that the code in `model.py` requires version 3.3 (or newer) of python. The specific versions of all packages and libraries mentioned in Section 4.7 were found to be compatible. It may be possible that breakages occur with other versions. The pyenv package may be required to ensure the correct version of python is available to the modelling code. Optionally, additional MILP solvers may be installed, such as those mentioned in Section 4.7.1.

- Clone the model repository by running the following from the terminal:

```
$ mkdir -p ~/git
$ cd ~/git
$ git clone https://github.com/multipitch/dissertation
```

- To make the `runmodel` executable available to the user, it must be appended to the user's `PATH` environment variable. This can be achieved, for the active terminal session, by running the following from within the terminal:

```
$ export PATH=$PATH:<location>
```

  where `<location>` is the path to the directory containing `runmodel` e.g.

```
$ export PATH=$PATH:~/git/dissertation/src
```

  The above redefinition of `PATH` can be made permanent by appending the command to e.g. the user's `~/.bashrc` file.

- To run using the default settings, given a directory containing valid input data (`buffers.csv`, `vessels.csv` and `parameters.ini` files), navigate to the directory and run the following from the command line:

  ```
  $ runmodel
  ```

  The above command uses PuLP's default solver (Cbc), without any arguments. It solves for both the primary and secondary models and generates equipment time utilisation plots for both, saving them to the directory as `plot1.pdf` and `plot2.pdf`.

- Several optional parameters may be passed to the `runmodel` executable, e.g. running

  ```
  $ runmodel -s CPLEX --no-secondary
  ```

  will run the model using the CPLEX solver and won't perform the secondary optimisation.

  A description of all input parameters is available by running `runmodel` with the `-h` or `--help` flags.

- Sample data is available in the examples directory of the repository. The command below gives an example of how to carry out a run on one of the sample data sets:

  ```
  $ runmodel -P  /git/dissertation/examples/random/
  ```

Program code description

Classes in `model.py`

The `model.py` file contains the following classes; a brief overview of each class is given in the paragraphs hereafter.

- `Parameters`
- `Vessels`
- `Buffers`
- `Variable`
- `Variables`
- `Results`

The `Parameters` class is used to store parameters data (see Section 3.4). It can read the data from a file, or be passed the data as a `dict`.

The `Vessels` class is used to store vessel data (see Section 3.2). Similar to the `Parameters` class, it can read the data from a file, or be passed the data as a `dict`.

The `Buffers` class is used to store buffer data (see Section 3.3). Again, it can read the data from a file, or be passed the data as a `dict`. It is also used to store some results when a problem has been solved; it contains a class method, `get_results` for this purpose.

The `Variable` class is used to define a variable, e.g. $\boldsymbol{z}_n$ or $\boldsymbol{w}_{nkp}$. From the point of view of the `pulp` module, a variable is a single scalar entity (a `pulp.LpVariable` object). An instance of the `Variable` class represents all the elements of a multidimensional variable e.g. $\boldsymbol{x}_{np}\ \forall n \in N,\ \forall p \in P$. The class is initialised with information on the multidimensional variable, such as dimensions, bounds and whether the variable is integer or real valued. When the model is solved, the `evaluate` class method is used to create a multidimensional array of the evaluated values of each scalar variable object.

The `Variables` class simply exists to contain a group of `Variable` class instances.

The `Results` class is a container for holding the solution variables. It has an `add` method for populating itself with data.

Key functions in `model.py`

The `model.py` file also contains the following functions:

- `csv_columns_to_dict_of_lists`
- `variable_iterator`
- `variable_iterator_loc`
- `_variable_iterator_loc`
- `unflatten_variable`
- `define_problem`
- `initial_objective`
- `secondary_objective`
- `generate_random_model`
- `run_primary`
- `run_secondary`
- `standard_run`

- `run_random_models`
- `many_random`
- `one_random`

The `csv_columns_to_dict_of_lists` function reads from csv files where the header row contains keys and subsequent rows contain values. It converts this data to a `dict`, with the header row entries as keys and the columns of data below a header as a list of values for that key. This is used to read from the `buffers.csv` and `vessels.csv` files.

The `variable_iterator` and `variable_iterator_loc` functions are recursive functions that may be used to access the components of a multidimensional variable in order. The former just returns the component, whilst the latter also returns its index. Both functions are wrappers to the function `_variable_iterator_loc`, which actually performs the recursion. These are required to flatten a multidimensional variable into a list so that it is suitable for passing to `pulp`.

The `unflatten_variable` variable function essentially performs the reverse of the above iterator functions; given a required set of dimensions and a list of objects, it unflattens the list into a multidimensional array of objects with the requisite dimensions. It calls `variable_iterator_loc` to do this, i.e. it also operates recursively.

The `define_problem` function implements all of the equations in Section 4.5.8, except for the objective function. It starts by initialising a problem instance (i.e. a `pulp.LpProblem` object), then defines all the variables used in the equations, then adds all the equations to the problem instance. The objective function is omitted because this allows the function to be used for both the primary and the secondary objective runs.

The `initial_objective` function sets the objective function as per equation 4.3.3. This is used, after `define_problem` to fully define the complete problem summarised in Section 4.5.8.

Similarly, the `secondary_objective` function sets the objective function as per equation 4.6.1 and also implements the additional constraint required by the secondary problem (equation 4.6.3). This function is used to define the secondary problem, as summarised in complete problem summarised in Section 4.6.2.

The remaining functions are used to generate models from defined or random data and to run the models via pulp.

Key functions in `plot.py`

The `plot.py` file contains a number of functions, listed below, for generating various plots. It uses the `matplotlib` library extensively.

- `single_cycle_plot`
- `explanatory_plot`
- `sched_plot_single`
- `sched_plot_all`
- `complexity_plot`
- `timing_plot`
- `cyclic_xranges`
- `read_durations`

The `single_cycle_plot` function produces an equipment time utilisation plot for a solved problem. Examples in this document include figures 4.3 and 4.4 in this chapter. This is the main visual output from a solved problem.

The `explanatory_plot` function produces the plot in Figure 3.1. The `sched_plot_single` and `sched_plot_all` functions are used to produce the plots in Figures 4.1 and 4.2 respectively. The `complexity_plot` function produces is used to produce the plots in Figures 6.1 and 6.2. These are all static, explanatory plots used to illustrate the report only.

The `timing_plot` function is used to generate a box-plot showing the duration taken to solve a random problem as a function of the number of buffers in the problem. Figure 5.2.1 is an example of such a plot.

The `cyclic_xranges` function is called by the `single_cycle_plot` function and the `sched_plot_all` function. If an operation overlaps the single cycle range boundary, it splits the operation into two segments, one from the start of the operation to T and the other from $t = 0$ to the end of the operation.

The `read_durations` function is used to read timing data from a file. This may be used by `timing_plot`. The capability to read and write duration data to a file is useful as it can take a considerable amount of time to obtain the duration data and if it was destroyed when the program terminates, tweaking the appearance of the plot would otherwise require re-running all the constituent simulations again.

Real-world data set 1

Table A1: Buffer data for real-world data set 1

names	required volumes U_n (l)	use start times $t^*_{USE,n}$ (h)	use durations $\Delta t_{USE,n}$ (h)
Buffer #1	19 676.0	733.87	48.99
Buffer #2	18 528.0	644.18	36.95
Buffer #3	17 346.0	684.60	49.27
Buffer #4	15 055.0	794.96	28.01
Buffer #5	13 896.0	628.54	52.59
Buffer #6	10 267.0	728.99	53.12
Buffer #7	10 070.0	677.94	20.86
Buffer #8	6797.0	728.99	53.12
Buffer #9	6716.0	612.90	70.40
Buffer #10	4632.0	645.50	35.63
Buffer #11	3780.0	729.34	43.19
Buffer #12	3694.0	678.92	42.17

Table A2: Vessel data for real-world data set 1

names	volumes V_m (l)	costs c_m (–)
2000 l	2222.0	95.64
4000 l	4444.0	144.96
5000 l	5556.0	165.72
8000 l	8889.0	219.71
10 000 l	11 111.0	251.19
12 000 l	13 333.0	280.83
15 000 l	16 667.0	320.37
20 000 l	22 222.0	380.73
25 000 l	27 778.0	435.28

Table A3: Global parameters for real-world data set 1

symbol	short description	value	unit
T	process cycle time	96.0	h
$\Delta t_{PREP,PRE}$	prep pre duration	11.0	h
$\Delta t_{PREP,POST}$	prep post duration	2.5	h
$\Delta t_{TRANSFER}$	transfer duration	2.2	h
$\Delta t_{HOLD,PRE}$	hold pre duration	9.0	h
$\Delta t_{HOLD,POST}$	hold post duration	1.0	h
$\Delta t_{HOLD,MIN}$	minimum hold duration	12.0	h
$\Delta t_{HOLD,MAX}$	maximum hold duration	60.0	h
$f_{MINFILL}$	minimum fill ratio	0.27	–
f_{UTIL}	maximum utilisation ratio	0.7	–

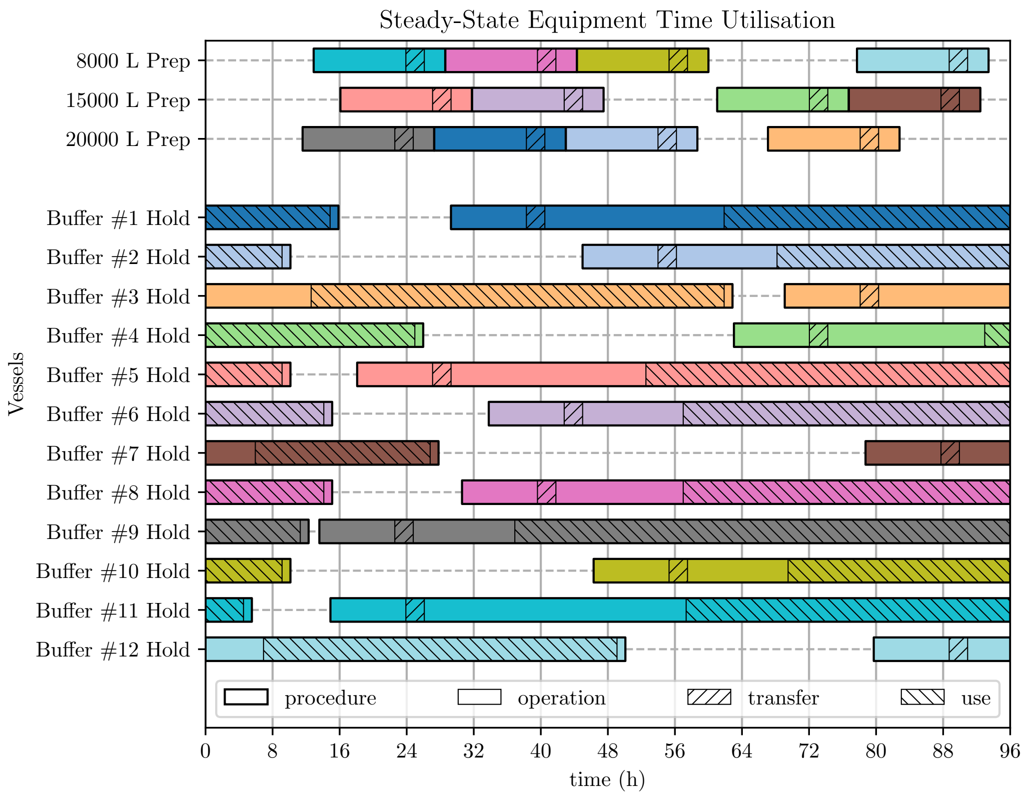


Figure A1: Real-world data set 1 – complete model with minimised hold times

Real-world data set 2

Table A4: Buffer data for real-world data set 2

names	required volumes U_n (l)	use start times $t^*_{USE,n}$ (h)	use durations $\Delta t_{USE,n}$ (h)
Buffer #1	72.0	93.75	4.86
Buffer #2	208.0	86.72	9.19
Buffer #3	242.0	36.57	3.51
Buffer #4	453.0	73.51	5.59
Buffer #5	478.0	83.76	4.95
Buffer #6	567.0	68.78	2.29
Buffer #7	618.0	28.01	1.89
Buffer #8	810.0	110.57	2.79
Buffer #9	845.0	110.21	31.68
Buffer #10	1107.0	112.46	37.65
Buffer #11	1203.0	98.62	1.62
Buffer #12	1639.0	66.43	22.29
Buffer #13	1772.0	110.93	39.19
Buffer #14	2128.0	26.06	4.05
Buffer #15	2171.0	66.97	21.75
Buffer #16	3873.0	36.66	48.39
Buffer #17	4503.0	44.42	35.76
Buffer #18	4987.0	94.05	12.97
Buffer #19	5826.0	103.81	47.39
Buffer #20	6225.0	111.65	32.39
Buffer #21	10 682.0	42.16	40.72
Buffer #22	14 253.0	99.16	13.63

Table A5: Vessel data for real-world data set 2

names	volumes V_m (l)	costs c_m (–)
100 l	100.0	15.85
200 l	200.0	24.02
500 l	500.0	41.63
1000 l	1000.0	63.10
2500 l	2500.0	109.34
5000 l	5000.0	165.72
7500 l	7500.0	211.37
10 000 l	10 000.0	251.19
15 000 l	15 000.0	320.37

Table A6: Global parameters for real-world data set 2

symbol	short description	value	unit
T	process cycle time	120.0	h
$\Delta t_{PREP,PRE}$	prep pre duration	10.3	h
$\Delta t_{PREP,POST}$	prep post duration	2.7	h
$\Delta t_{TRANSFER}$	transfer duration	2.1	h
$\Delta t_{HOLD,PRE}$	hold pre duration	7.0	h
$\Delta t_{HOLD,POST}$	hold post duration	2.2	h
$\Delta t_{HOLD,MIN}$	minimum hold duration	1.0	h
$\Delta t_{HOLD,MAX}$	maximum hold duration	60.0	h
$f_{MINFILL}$	minimum fill ratio	0.3	–
f_{UTIL}	maximum utilisation ratio	0.8	–

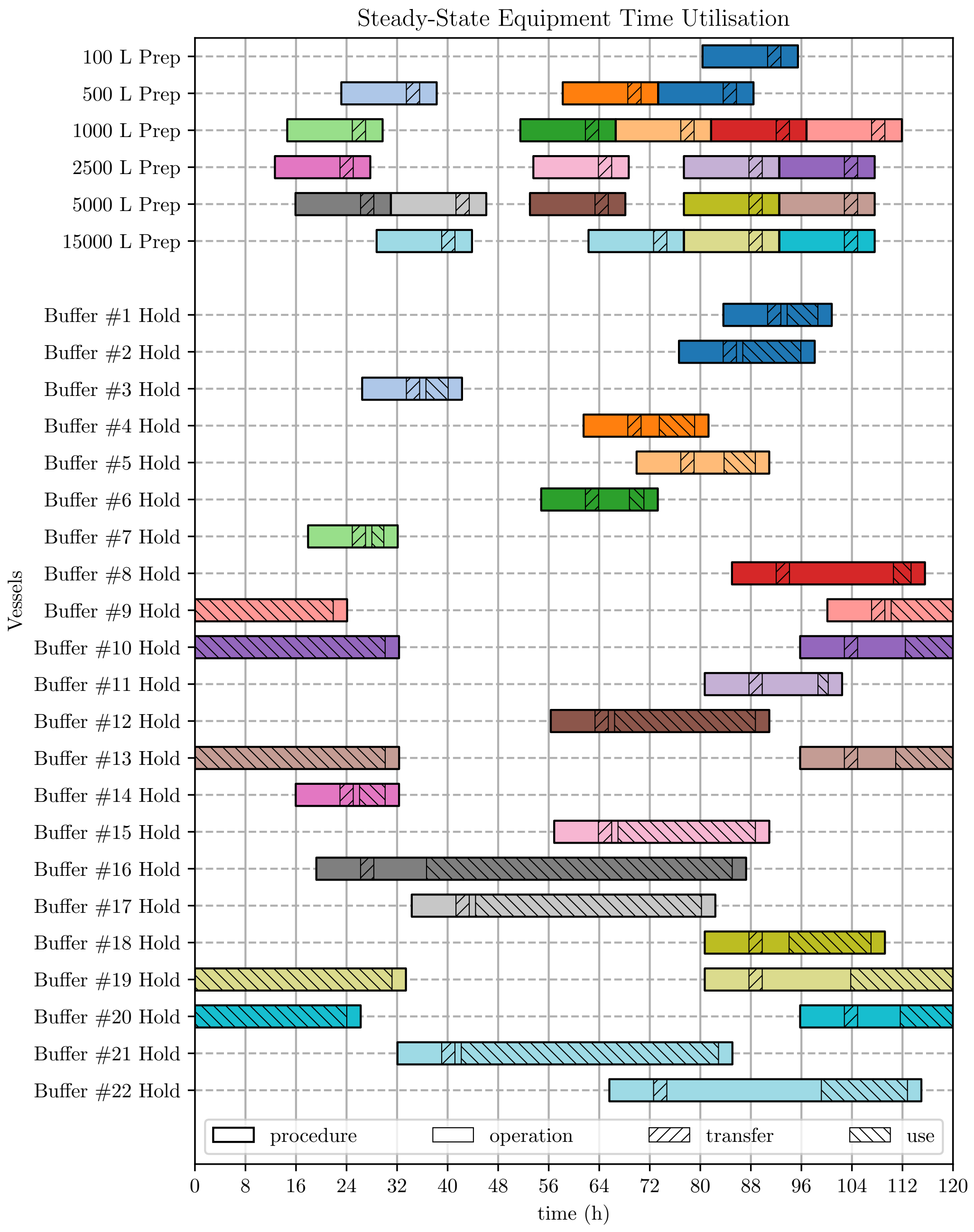


Figure A2: Real-world data set 2 – complete model with minimised hold times

Bibliography

Ahmed, S. and N. V. Sahinidis. 2000. Analytical investigations of the process planning problem. *Computers & Chemical Engineering*, **23**(11–12): 1605–1621. doi:10.1016/S0098-1354(99)00312-9.

Al-Otaibi, G. A., M. D. Stewart *et al.*. 2004. Simulation model determines optimal tank farm design. *Oil & gas journal*, **102**(7): 50–55.

Bettinelli, A., A. Ceselli and G. Righini. 2010. A branch-and-price algorithm for the variable size bin packing problem with minimum filling constraint. *Annals of Operations Research*, **179**(1): 221–241. doi:10.1007/s10479-008-0452-9.

Cavin, L., U. Fischer, F. Glover and K. Hungerbühler. 2004. Multi-objective process design in multi-purpose batch plants using a tabu search optimization algorithm. *Computers & Chemical Engineering*, **28**(4): 459–478. doi:10.1016/j.compchemeng.2003.07.002.

Cavin, L., U. Fischer, A. Mošať and K. Hungerbühler. 2005. Batch process optimization in a multipurpose plant using tabu search with a design-space diversification. *Computers & Chemical Engineering*, **29**(8): 1770–1786. doi:10.1016/j.compchemeng.2005.02.039.

Dantzig, G. B. 1957. Discrete-variable extremum problems. *Operations Research*, **5**(2): 266–277. doi:10.1287/opre.5.2.266.

Dedieu, S., L. Pibouleau, C. Azzaro-Pantel and S. Domenech. 2003. Design and retrofit of multiobjective batch plants via a multicriteria genetic algorithm.

Computers & Chemical Engineering, **27**(12): 1723–1740. doi:10.1016/S0098-1354(03)00155-8.

Dietz, A., A. Aguilar-Lasserre, C. Azzaro-Pantel, L. Pibouleau and S. Domenech. 2008. A fuzzy multiobjective algorithm for multiproduct batch plant: Application to protein production. *Computers & Chemical Engineering*, **32**(1–2): 292–306. doi:10.1016/j.compchemeng.2007.05.011.

Harjunkoski, I., C. T. Maravelias, P. Bongers, P. M. Castro, S. Engell, I. E. Grossmann, J. Hooker, C. Méndez, G. Sand and J. Wassick. 2014. Scope for industrial applications of production scheduling models and solution methods. *Computers & Chemical Engineering*, **62**: 161–193. doi:10.1016/j.compchemeng.2013.12.001.

Knuth, D. E. 1976. Big omicron and big omega and big theta. *SIGACT News*, **8**(2): 18–24. doi:10.1145/1008328.1008329.

Korte, B. and J. Vygen. 2012. *Combinatorial Optimization: Theory and Algorithms*. Springer-Verlag, Berlin, Heidelberg. doi:10.1007/978-3-642-24488-9.

Martello, S. and P. Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA. ISBN 0-471-92420-2.

Otto, R., A. Santagostino and U. Schrader. 2014. Rapid growth in biopharma: Challenges and opportunities. online. Accessed 8th June 2016. URL `http://www.mckinsey.com/industries/pharmaceuticals-and-medical-products/our-insights/rapid-growth-in-biopharma`

Petrides, D., D. Carmichael, C. Siletti and A. Koulouris. 2014. Biopharmaceutical process optimization with simulation and scheduling tools. *Bioengineering*, **1**(4): 154–187. doi:10.3390/bioengineering1040154.

Schaschke, C. 2014. *A Dictionary of Chemical Engineering*. Oxford University Press, Oxford. doi:10.1093/acref/9780199651450.001.0001.

Sharda, B. and A. Vazquez, 2009. Evaluating capacity and expansion opportunities at tank farm: a decision support system using discrete event simulation. In: *Simulation Conference (WSC), Proceedings of the 2009 Winter*, pages 2218–2224. IEEE.

Stewart, M. D. and L. D. Trierwiler. 2005. Simulating optimal tank farm design. *PTQ Magazine*, (2).

Taha, H. A. 2017. *Operations Research: An Introduction.* Global Edition. Pearson, Harlow, UK, 10th edition. ISBN 9781292165547.

Terrazas-Moreno, S., I. E. Grossmann and J. M. Wassick. 2012. A mixed-integer linear programming model for optimizing the scheduling and assignment of tank farm operations. *Industrial & Engineering Chemistry Research*, **51**(18): 6441–6454. doi:10.1021/ie202217v.

Toumi, A., C. Jurgens, C. Jungo, B. Maier, V. Papavasileiou and D. Petrides. 2010. Design and optimization of a large scale biopharmaceutical facility using process simulation and scheduling tools. *Pharmaceutical Engineering*, **30**(2): 1–9.

List of Notation

Symbol	**Description**	**Ref**
$\boldsymbol{a}_{nk}$	buffers n and k prepared in same vessel (binary)	4.5.3
c_m	(relative) cost of vessel m	3.2
f_{MAXUSE}	buffer maximum use duration ratio	3.5.1
$f_{MINFILL}$	vessel minimum fill ratio	3.4
f_{MINUSE}	buffer minimum use duration ratio	3.5.1
f_{UTIL}	preparation slot maximum utilisation ratio	3.4
k	secondary buffer index ($k \in \mathcal{N}, k \neq n$)	4.5.2
m	vessel size index ($m \in \mathcal{M}$)	3.2
n	buffer index ($n \in \mathcal{N}$)	3.3
p	slot index ($p \in \mathcal{P}$)	4.2
$\boldsymbol{q}_n$	the buffer n hold operation crosses the single-cycle boundaries (binary)	4.5.5
$\boldsymbol{r}_n$	$\boldsymbol{t}_{LOWER,n}$ occurs before $\boldsymbol{t}_{PREP,n}$ in the single-cycle window (binary)	4.5.6
$\boldsymbol{s}_n$	$\boldsymbol{t}_{UPPER,n}$ occurs after $\boldsymbol{t}_{PREP,n}$ in the single-cycle window (binary)	4.5.6
$\boldsymbol{t}_{LOWER,n}$	lower bound of feasible scheduling region for all buffers $k > n$ with respect to buffer n	4.5.6
$\boldsymbol{t}_{PREP,k}$	preparation reference time for buffer k	4.5.5
$\boldsymbol{t}_{PREP,n}$	preparation reference time for buffer n	4.5.5
$\boldsymbol{t}_{UPPER,n}$	upper bound of feasible scheduling region for all buffers $k > n$ with respect to buffer n	4.5.6
$t_{USE,n}$	buffer n time of first use, normalised	3.3

Symbol	**Description**	**Ref**
$t^*_{USE,n}$	buffer n time of first use, un-normalised	3.3
Δt_{FEAS}	maximum feasible buffer use duration	3.5.1
$\Delta t_{HOLD,MAX}$	maximum allowable buffer hold duration	3.4
$\Delta t_{HOLD,MIN}$	minimum allowable buffer hold duration	3.4
$\Delta t_{HOLD,POST}$	duration of post-use operations in buffer hold procedures	3.4
$\Delta t_{HOLD,PRE}$	duration of operations prior to receiving buffer in buffer hold procedures	3.4
Δt_{PREP}	total duration of buffer preparation procedures	4.4.4
$\Delta t_{PREP,PRE}$	duration of operations prior to transferring out buffer in buffer preparation procedures	3.4
$\Delta t_{PREP,POST}$	duration of operations post transferring out buffer in buffer preparation procedures	3.4
$\Delta t_{USE,n}$	duration of use of buffer n	3.3
$\Delta t_{TRANSFER}$	duration of transfers from buffer preparation vessel to buffer hold vessel	3.4
$\boldsymbol{u}_n$	feasible scheduling window for buffer k with respect to buffer n does not cross cycle boundary (binary)	4.5.6
$\boldsymbol{v}_{nk}$	feasible scheduling window for buffer k with respect to buffer n occurs before buffer n preparation procedure (binary)	4.5.6
$\boldsymbol{w}_{nkp}$	distinct buffers n and k are both made in slot p (binary)	4.5.3
$\boldsymbol{x}_{np}$	buffer n is prepared in slot p (binary)	4.4.1
$\boldsymbol{y}_{mp}$	a vessel of size m is in slot p (binary)	4.3
$\boldsymbol{z}_n$	buffer n hold duration	4.5.1
M	number of vessel sizes	3.2
$\mathcal{M}$	set of vessel sizes	3.2
N	number of buffers	3.3
$\mathcal{N}$	set of buffers	3.3
P	number of slots	4.2
$\mathcal{P}$	set of slots	4.2

Symbol	**Description**	**Ref**
T	process cycle time (start–to–start duration)	3.4
U_n	volume of buffer n to be prepared	3.3
V_m	maximum working volume of vessel size m	3.2
V_{MAX}	largest maximum working volume of available vessel sizes	4.4.3
$\boldsymbol{Y}$	secondary objective; sum of buffer hold times, given minimal total vessel cost	4.6.1
$\boldsymbol{Z}$	primary objective; total vessel cost	4.3
Z'	minimal total vessel cost	4.6.1

Notes:

Decision variables and objective function values are in ***bold*** text.

Subscripts in lower-case represent indices. Subscripts in upper-case are descriptive. Both subscript types may be used simultaneously, e.g. in the case of $t_{USE,n}$.

A prime superscript denotes an optimum value, e.g. Z'.

Lower-case Greek letters are used throughout the text as 'throwaway' variables, to explain certain concepts and are not included in the list of notation.