UNIVERSITY OF THE WITWATERSRAND

DOCTORAL THESIS

Metaheuristic Approaches for Scheduling of Multipurpose Batch Plants

Author:

Matthew John WOOLWAY

Supervisor:

Prof. Thokozani MAJOZI

WITS UNIVERSITY

A thesis submitted in fulfilment of the requirements

for the degree of Doctor of Philosophy

in the

School of Chemical and Metallurgical Engineering

June 26, 2018

Declaration of Authorship

I, Matthew John WOOLWAY, declare that this thesis titled, "Metaheuristic Approaches for Scheduling of Multipurpose Batch Plants" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: M. Woolway

Date: 26/06/2018

UNIVERSITY OF THE WITWATERSRAND

Abstract

Faculty of Engineering and the Built Environment

School of Chemical and Metallurgical Engineering

Doctor of Philosophy

Metaheuristic Approaches for Scheduling of Multipurpose Batch Plants

by Matthew John WOOLWAY

The field of batch chemical process has seen a significant rise in research over the last five decades as changes in the economic climate have lead to an increased demand for the manufacturing of high-value small-volume products. Due to the dependency on time, batch processes are considerably more complex than their continuous process counterparts. The predominant approach in batch process literature makes use of mathematical programming, whereby binary variables are utilised to indicate the assignment of certain tasks to capable units. This mathematical programming strategy, coupled with the aforementioned time complexity can lead to computational intractability due to the extended enumeration of binary variables. In this thesis, the reduction of computational time required in the solution of multipurpose batch plant scheduling is considered.

Due to the infeasible computational times required to solve mathematical programming models in multipurpose batch plant scheduling, often close-to-optimal solutions rather than global optimal solutions are accepted. If close-to-optimal solutions are acceptable then it is reasonable to explore non-deterministic metaheuristic strategies to reduce the required computational time. In order

to apply these strategies, generalised frameworks consistent with metaheuristic approaches are necessary. Presently, no decoupled generalised framework suitable for various metaheuristic implementation exists in the literature.

As a result, this thesis presents two novel mathematical frameworks for the representation of batch scheduling. Specifically, one framework for discrete-time approaches and another framework for continuous-time approaches. In each framework, two well-known literature examples are considered. In addition, three metaheuristic techniques are applied to these literature examples, namely, genetic algorithms (GA), simulated annealing (SA) and migrating bird optimisation (MBO). The resultant framework allows for experimentation of 12 variants of the literature examples to be investigated, which can be compared to the currently accepted mixed integer linear programming (MILP) approach.

In the aforementioned experiment, simulated results with the metaheuristics implemented under the newly introduced frameworks showed a reduction in computational time of up to 99.96% in the discrete-time approach and 99.68% in the continuous-time approach. Additionally, the genetic algorithm showed to be the best performer of the metaheuristic suite, often obtaining the global optimum in short-time horizons and close-to-optimal solutions in the medium-to-long time horizons. Furthermore, parallel implementations were explored and showed additional time reduction would be possible, with certain workloads terminating 2 orders of magnitude less in computational time than serial implementations.

The results show the application of metaheuristics to the scheduling of multipurpose batch plants are indeed appropriate and are able to obtain close-to-optimal solutions to that of their MILP counterparts at considerably reduced computational times.

Acknowledgements

I would like to thank my thesis supervisor Professor Thokozani Majozi for his guidance and mentorship.

I would also like to thank my collegues, especially Dr Van Zyl, Professor Celik, Dr Hutchinson and Ms Macnamara for all their valuable input.

Finally, thank you to my family and friends for all of their support during this process.

Publications

Aspects of this work have been submitted as follows:

M. Woolway and T. Majozi. “A Novel Metahueristic Framework for the Scheduling of Multipurpose Batch Plants". *Chemical Engineering Science, 2018*. In review.

Contents

List of Figures

List of Tables

List of Abbreviations

MILP	**M**ixed **I**nteger **L**inear **P**rogram
MINLP	**M**ixed **I**nteger **N**on**l**inear **P**rogram
S&M	**S**eid and **M**ajozi
GA	**G**enetic **A**lgorithm
SA	**S**imulated **A**nnealing
MBO	**M**igrating **B**ird **O**ptimisation
TSP	**T**ravelling **S**alesman **P**roblem
QAP	**Q**uadratic **A**ssignment **P**roblem
KP	**K**napsack **P**roblem
MU	**M**onetary **U**nits
HCF	**H**ighest **C**ommon **F**actor

Nomenclature

Indices

f, f'	product family
i, i'	batch task
j, j'	batch processing unit
k	time slot
l	stage
n, n'	event point
n_{last}	final event point
s	state
t, t'	time point

Sets

I	all tasks
I_j	tasks that can be performed in unit j
$I_j^f, I_j^{f'}$	tasks for family f and f' that can be performed in unit j respectively
I_s^c, I_s^p	tasks which consume and produce state s respectively
J	all units
L	all stages
N	all event points

S	all material states
T	all time points
X	all CPU cores

Parameters

α_{ij}	fixed processing time of task i in unit j
β_{ij}	linear coefficient for variable processing time of task i in unit j
ϵ	tolerance value for temperature in simulated annealing
η	the number of tours to perform in migrating bird optimisation
γ	the number of time periods per hour according to the HCF
κ	the number of neighbouring solutions to generate for migrating bird optimisation
ν	cooling rate of the simulated annealing
ψ	rate of mutation for genetic algorithm
ρ^c_{is}, ρ^p_{is}	amount of state s consumed and produced by task i respectively
$\tau_{iff'}$	time required to clean unit j when switching from product family f to f'
$\tau_{ji'i}$	time required to clean unit j when switching from task i to i'
$\tau_{jff'}$	time required to clean unit j when switching from batch i' to i
θ	rate of crossover for genetic algorithm

ξ	the number of neighbouring solutions to be shared to next solution for migrating bird optimisation
c_p	the crossover point for crossover to be applied within the GA
C_s	storage capacity for state s
D_{st}	delivered amount of state s at time point t
H	time horizon
i_m	operations instruction matrix
K	maximum number of iterations for migrating bird optimisation
LM	length of Markov chain for simulated annealing
m_p	the time interval vector element on which mutation within the GA is to be applied to
M	a arbitrarily large positive value
P	population size of trial solutions for metaheuristic application
q	size of initial population for migrating bird optimisation
t_v	time interval vector for metaheuristics
TM	starting temperature for simulated annealing
R_{st}	received amount of state s at time point t
$V_{ij}^{\max}$	max capacity of unit j performing task i
$V_{ij}^{\min}$	min capacity of unit j performing task i
x	number of CPU cores to utilise

Binary Variables

W_{ijt}	determines if task i starts in unit j at time point t

W_{ijkl}	determines if stage l of task i is assigned to time slot k of unit j
$wv(i,n)$	determines if task i begins at event point n
$yv(j,n)$	determines if unit j is used at event point n

Continuous Variables

B_{ijt}	amount of material beginning task i in unit j at time point t
$B(i,j,n)$	amount of material beginning task i in unit j at event point n
$D(s,n)$	amount of state s delivered at event point n
$ST(s,n)$	amount of state s stored at event point n
$T^s(i,j,n), T^f(i,j,n)$	start and end times of task i in unit j at event point n
Tsi_{il}, Tei_{il}	start and end time of stage l in order i

Chapter 1

Introduction

1.1 Background

The modern world has seen an economic boom over the last half-century and with it the need for manufacturers to constantly expand and increase their operation in size and complexity. The primary objective of most companies and organisations is to increase profitability by maximising cost reduction and remaining competitive in a free market economy. These coveted improvements are achievable with efficient optimisation of the allocation of resources. As a result, production scheduling techniques have been identified as a means of increasing productivity and resource utilisation. The role of scheduling in all forms of production becomes increasingly important as society demands greater variety, resulting in a decrease of product life-spans and the need for the development of new technologies. The chemical process industry is no exception; examples can easily be found in batch processes, polymers and refinery operations etc. Historically, these problems have been tackled and modelled using mathematical programming techniques.

This research considers the scheduling of batch plants. Batch processes have only undergone extensive research over the last five decades. This is in part due to the shift in requirement from global markets. Prior to the middle of the twentieth century, most production constituted of mass production and large volume order

books. This lead to a preference towards continuous processes. Entering the 1970s, however, markets changed. Market volatility required processes which were easily adaptable to abrupt changes. As a result, optimal schedules became increasingly important in order to minimise any feasible profit loss. These minimisations can be measured with a number of different metrics and have been the aim of researchers in the field of batch scheduling. Common metrics include profit maximisation, cost minimisation, makespan reduction, minimising tardiness, or others unique to specific manufacturers. Industrial engineers have studied the field of scheduling extensively; however, the nuances of chemical plants imply that often these lessons and techniques cannot simply be applied. In general, the constraints and problem description found within a chemical plant is very different. Industrial engineering and supply chain more often than not, consider a discrete set of jobs to be performed by machines or humans. Intrinsically, this is easy to understand: an item is cut on Machine A and then cut another way on Machine B, for example. A chemical plant may be subject to vastly more difficult constraints, for example: a batch of one chemical may finish on one reactor but only a limited amount can continue to the next. Instantly, this can cause issues, perhaps the chemical in question cannot stay in its present state longer than a set time, or perhaps more cannot be mixed with it until the present machine is void of all volume. In addition, the units themselves present concern. Often a machine may require special cleaning after every batch to avoid impurities, for example. The examples above list just a small subset of cases that may arise within a chemical plant. Thus, unique approaches to chemical batch plants are required.

The dichotomy of batch processes consist of multiproduct batch plants and multipurpose batch plants. Multiproduct batch plants are analogous to flowshop problems in other fields. Specifically, there is a fixed configuration of units that materials flow through to produce the final product. Multipurpose batch plants

pertain to the scenario, where given a recipe for some product(s), different units are able to produce these final product(s), meaning a fixed configuration is not set.

1.2 Motivation

The seminal papers in the field of multipurpose batch plant scheduling from a chemical engineering perspective, were published in the early 1990's. These contributions and indeed more recent publications, have considered these problems through discrete and continuous approaches. The major drawback across these approaches is that of computational speed (Majozi et al., 2017). Regardless of whether the approach is discrete or continuous, the formulations all belong to mixed integer formulations and thus suffer the inevitability of NP-hardness, implying that the existence of polynomial time[1] solutions is unknown (Garey and Johnson, 1979). Whilst short-term scheduling is very achievable in polynomial time, the idea of finding optimal solutions to schedules spanning a week or more are non-existent on account of the inescapable enumeration. This is the area which this research aims to investigate. Considering the massive increase in computational performance over the last 20 years, perhaps an approach which can exploit the scalability of this increase in performance may yield more success in the instances of medium-to-long time horizon scheduling. This is done through the use of metaheuristics. Although these techniques do not attempt to always find global solutions unlike their integer program counterparts, the ability to find close-to-optimal solutions in polynomial time seems more advantageous than not finding a solution at all. The research here, introduces two novel generalised decoupled frameworks suitable for standard metaheuristic implementation. In keeping with the present literature, one formulation is a discrete-time approach and the second is a continuous-time approach.

[1] The running time is upper bounded by some polynomial expression in the size of the input of the problem (Sipser, 2005).

1.3 Research Objectives

Formally, the objectives of this research are:

- To develop a novel framework for multiproduct and multipurpose batch plants that relevant metaheuristic techniques can holistically be applied to. This includes both a discrete-time approach and a continuous-time approach.
- To enable the metaheuristic framework just described, to provide close-to-optimal solutions to the scheduling of multiproduct and multipurpose batch plants in substantially reduced computational time when compared to the current approaches in the literature.
- To extend the above two points with an investigation in the parallel processing possibilities and its tractability as a means of computation in the field of multiproduct and multipurpose batch plant scheduling.

1.4 Thesis Structure

The remainder of this thesis has the following structure. Chapter 2 is a literature review considering publications making a key contribution, which are covered and explored throughout. Chapter 3 introduces the optimisation theory required from the metaheuristic standpoint and all other necessary information needed for the new formulations. It also discusses the problem description of the literature examples used and how the metaheuristics are applied to these examples. Chapter 4 contains results corresponding to two well-known literature examples. These results consider various cases across all implemented methods and collate into an analysis of performance under certain metrics. Chapter 5 draws conclusions from the results, and recommendations on future work is provided. All relevant appendices follow Chapter 5, with all references appearing at the end of the thesis.

Chapter 2

Literature Review

As mentioned in Chapter 1, the problem of short-term scheduling of both multiproduct and multipurpose batch plants has, in recent years, received focussed attention from industry and academia alike. This is in part, due to the increase in demand for high-value, small-volume products. This literature review does not attempt to provide a holistic description of the field, as several excellent reviews such as these can already be found in Reklaitis (1996), Pekny and Reklaitis (1998), Pinto and Grossmann (1998), Shah (1998), Kallrath (2002), Floudas and Lin (2004), Méndez et al. (2006), Shaik et al. (2006), Majozi (2010) and Majozi et al. (2017). The aim here is to provide a fundamental base of knowledge covering the current theory. This review serves to draw comparisons of the current methods found in literature to that of the new methodologies introduced herein.

A general statement to the description of scheduling problems is: the means of devising a decision-making processes which determines, the when, where and how, with regard to production of a collection of products. The production is done over set time horizons and is constrained by limited resources and processing recipes. This implies numerous decision variables involved, yielding a combinatorial quagmire. It is known that these problems are at least as hard as those found in Garey and Johnson (1979) thus rendering them within the set of NP-complete problems. This has clear implications with regard to the solutions of these problems. Since they are not bound

to polynomial time, a relatively small increase in problem size can, and will, lead to a surge in computational time. In addition, current deterministic formulations (such as those discussed here) suffer this intractability immensely due to the heavy reliance on binary variables.

Since process scheduling problems are affected by combinatorial nuances, it is important that effective mathematical formulations be developed to model the actual processes, while being adept at mitigating these computational restrictions. Much of the literature thus far finds itself in a subset of two main groups founded on the basis of time representation. Initially, there was only a sole representation, that of discrete time as introduced by Kondili et al. (1993). Here, time horizons in question were discretised into a number of equal time intervals. Inherently, this approach suffers inaccuracy, as the move from a continuous time horizon to a discrete one, invariably allows for suboptimal solutions. Furthermore, the increased number of binary variables introduced due to programming techniques lead to intractable enumeration. Attempts to overcome these issues, led to the development of continuous-time representations. Most recent research focuses on this approach as, theoretically, it should allow for the development of more efficient models and solution techniques: the cost for this coming at the expense of more complicated mathematical models.

This chapter is laid out as follows. Firstly, a description of the important components and parameters comprising a model are explored. Secondly, the discrete-time and continuous-time approaches are covered respectively.

2.1 Components of Scheduling Formulations

2.1.1 The Representation of Time

The fundamental issue of process scheduling problems is the representation of time. Literature is grouped into two main categories; discrete and continuous-time models.

Seminal work done by Kondili et al. (1993) made use of the discrete-time approach in the modelling process, as seen in Figure 2.1. With the time horizon dissected into a given number of uniform domains, events (such as the starting point, or end of a task) are confined within the boundaries of these time intervals. For an approximation of any merit, the size of each time internal needs to be suitably small. Generally, this is done through discretising time intervals into the highest common factor (HCF) of all processing times associated to units. The downside of this approach is that massive enumeration is encountered when the interval length is small relative to the time horizon. This is a major potential drawback, as large increases into computational power do little to mitigate the time required to obtain a solution, due to the sheer amount of computation required.

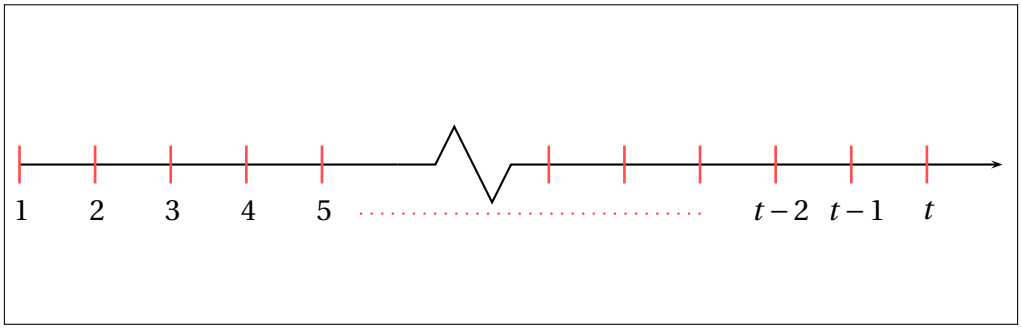

FIGURE 2.1: Discrete-time Representation (where t is the number of time points)

Due to the combinatorial issue arising with discrete-time models, continuous-time models were developed. In these models, events are allowed to take place at any point within the time domain. This was achieved by utilising variable event times,

which could take place across the entire process, or even specific units. An example of a continuous-time domain can be found in Figure 2.2. The leading idea behind continuous-time scheduling is to remove large quantities of enumeration by voiding the inactive event-time intervals found in its discrete-time counterpart. This in turn implies a much smaller integer program to solve and thus a “significantly" reduced computational duration. This new framework, however, comes at the cost of modelling difficulty, with potentially more ingenuity required than discrete model development. This approach is not without its own flaws either, as often, it is not actually known how many events points would be required to find an optimal solution for a given time horizon. Both these approaches will be discussed in further detail in Sections 2.2 and 2.3 respectively.

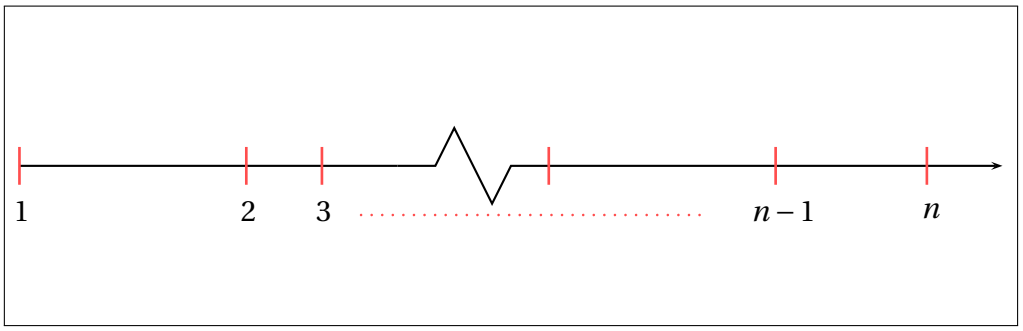

FIGURE 2.2: Continuous-time Representation (where n is the number of event points)

2.1.2 Process Scheduling Problems: Attributes and Parameters

Although we can consider the time representation as the parent nodes of a process scheduling problem formulation, there are also numerous other characteristics involved in the problem. Most of these are independent of the time representation used. These properties manifest themselves in various combinations from problem to problem. Here we attempt to cover the prominent factors found.

Process Topology

Firstly, we consider the types of processing sequences available to produce the end products. These form a dichotomy of sequential processes and network-represented processes, a breakdown of which features in Figure 2.3.

Sequential Processes: in cases where products follow the same processing sequence, *reaction-drying-packing*, for example, the ability to determine processing stages becomes available. Within each stage, units are not limited serially, but can in fact, be comprised of parallel units. Since batches represent production in these processes, the consideration of mass balances is moot.

Network-Represented Processes: more commonly, plant production recipes are intricate with different products rarely sharing recurrent recipes. As a result, processing networks are used to describe the production sequences. These networks are also consummate when handling batches which merge and/or split something enforcing the inclusion of material balances. The earliest network-representation development was by Kondili et al. (1993), where they introduced the general framework of the State-Task Network (STN). The STN is composed of two nodes, the **state**, describing raw materials, intermediate materials or end products. As seen in Figure 2.4, these nodes are represented with circles. **Task** nodes on the other hand, denoted by rectangles in Figure 2.4, describe an operation of some kind. In instances where a task consumes, merges or splits a state fractional, it is represented either through a percentage found on an edge, as on Figure 2.4, or as some ratio normalised to one (i.e. decimal value). A subsequent approach, developed by Pantelides (1994), went further than the STN, by introducing the Resource-Task Network (RTN), seen in Figure 2.5. This framework discusses storage, processing

units, material transfer and important utilities in a holistic manner. Finally, Majozi and Zhu (2001), introduced the State-Sequence Network (SSN). The SSN representation is solely based on states, removing the implementation of *tasks* and *units*. A major advantage of the SSN is its ability to reduce the number of binary variables used in the model, and thus, through solving mixed integer problems, reduced CPU times. An example is illustrated in Figure 2.6.

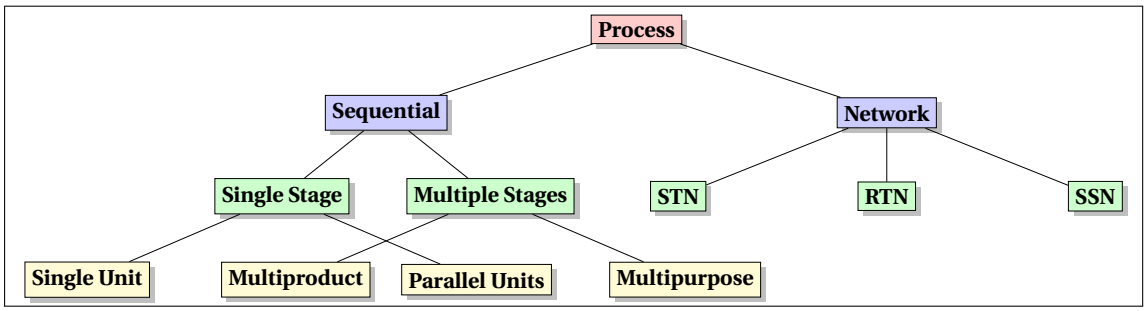

FIGURE 2.3: Process Sequences

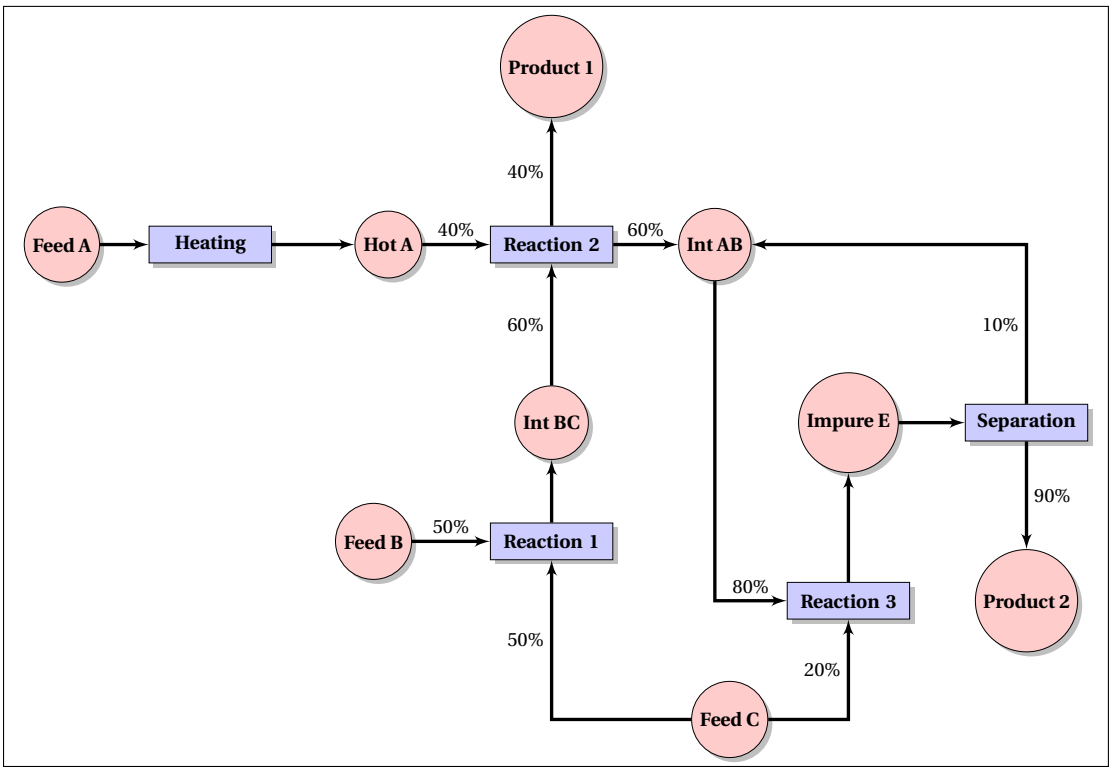

FIGURE 2.4: State-Task Network (STN)

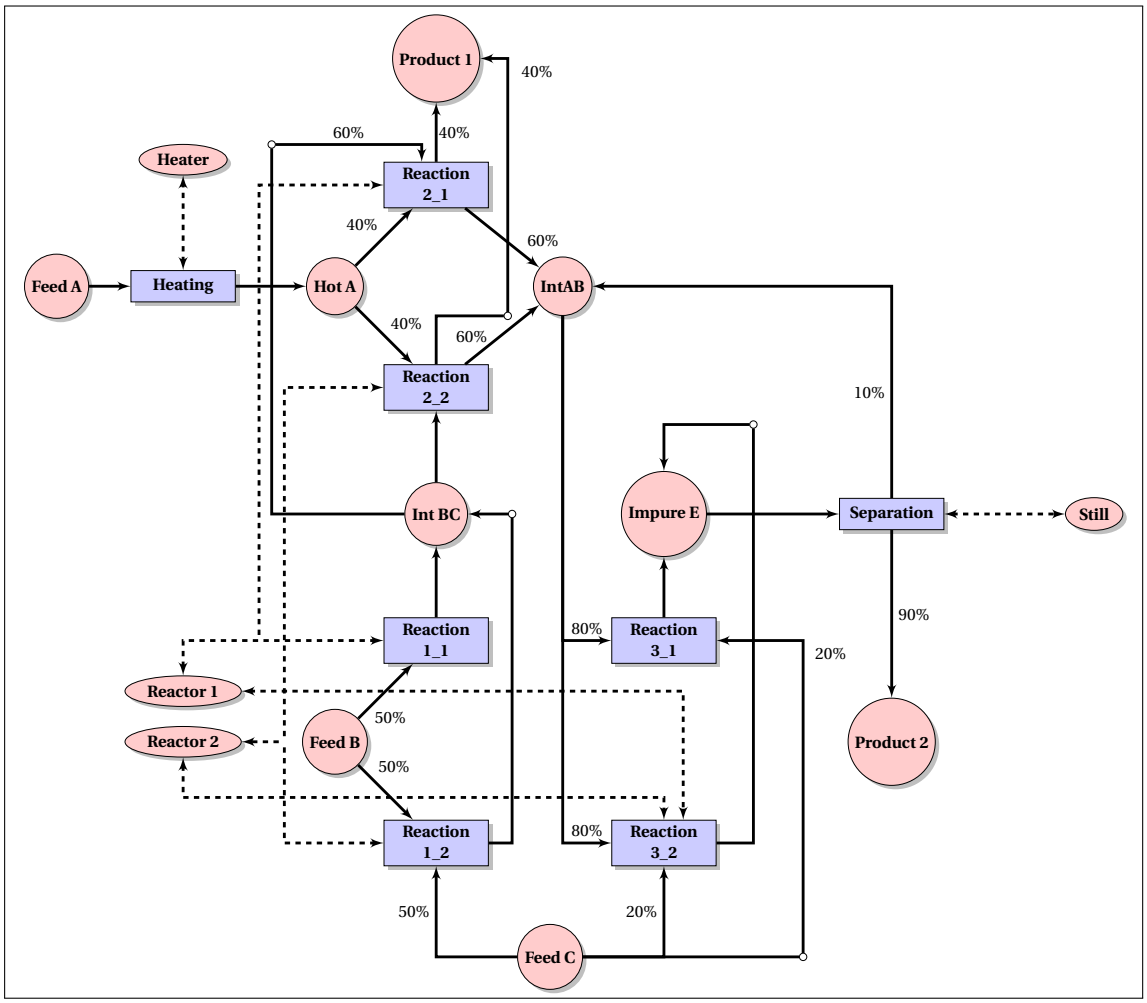

FIGURE 2.5: Resource-Task Network (RTN)

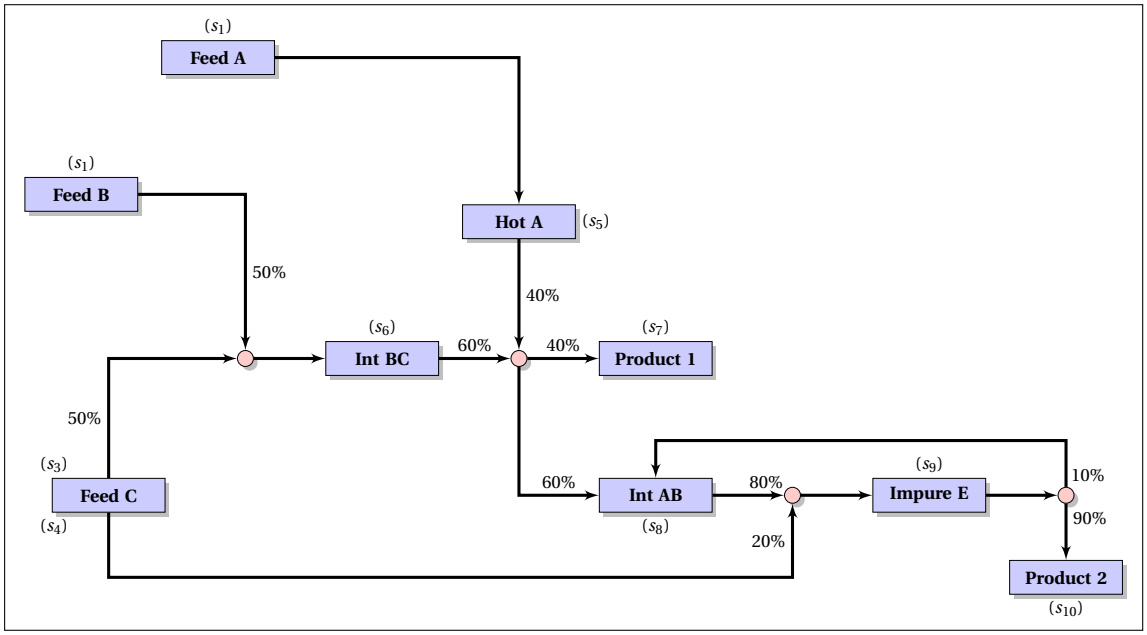

FIGURE 2.6: State-Sequence Network (SSN)

Storage States

The majority of storage (intermediate) situations subscribe to one of five main categories:

1. **Unlimited Intermediate Storage (UIS):** in situations arising with unlimited intermediate storage, the model does not need to take these states into account. This allows for a much simpler model.

2. **No Intermediate Storage (NIS):** cases such as these do not provide storage tanks for materials in the intermediate stage. Models make use of pseudo storage as material may remain in a processing unit (post task) before being moved to the next unit.

3. **Finite Intermediate Storage (FIS):** generally, for most plants, this is the likely case. Intermediate storage tanks exist between tasks. This, however, implies potential bottlenecks, for example, smaller capacity storage tanks between larger capacity processing units. Bottlenecks such as these will be covered in more detail when dealing with the literature examples to follow.

4. **Zero-Wait (ZW):** certain plants may be dealing with intermediate materials that need to be transferred to subsequent tasks instantaneously. This may be due to the chemical processes involved, or even the plant design.

5. **Finite-Wait (FW):** similar to that of Zero-Wait, intermediate material(s) needs to be processed within a short time after the preceding step finishes.

Model Objectives

The entire principle of Optimisation in research and engineering is maximising or minimising some objective(s). Process Scheduling problems are no different and following objectives akin to scheduling problems found in other branches of

engineering. Candidate objectives, for which solutions are aimed to be found, generally include:

- **Maximising Profit:** this is the most common of all. Under the equipment and resource constraints involved, here the objective is to find the schedule which will yield the largest profit margin in a given time horizon.
- **Minimising Makespan:** the optimal schedule under the same constraints listed above which completes the process the fastest is the objective. This objective is more deadline driven, it is not guaranteed to promote profit.
- **Minimising Tardiness and Earliness:** this schedule is most likely to meet clientèle deliverables. Similarly under the same constraints, this objective aims to find the optimal schedule which minimises deviations from due dates.
- **Minimising Cost:** given the similar constraints, an objective could be to minimise the overall cost incurred during the time horizon. Reasons for choosing this objective over profit maximisation could be the length of the time horizon, for example.

Miscellaneous Concerns

There are indeed numerous other concerns with regard to the process scheduling problem, however, as mentioned the aim of this research is not to provide comprehensive review. It is prudent though to, at least, mention some of these concerns briefly.

When developing these models, one should ensure that designs take into account the present state of the local environment. South Africa, for example, finds itself one of the scarcest water and power countries in the world. With that in mind, models should always factor in resource considerations. These generally present themselves as renewable resources, such as steam, cooling water and electricity.

Dependent upon the industry, different demands may be required. A model must therefore decide if there is demand for products solely at the end of the time horizon. Alternatively, a model should cater for situations when demand for products take place within a time horizon at some given points. These dates could be *a priori* or otherwise.

Plants must also deal with the operation of tasks. Set-ups may include a discrete batch task, where a given amount of material enters at the beginning of a task. Conclusion of the task leads to products produced. Continuous tasks are also possible. Materials would enter continuously into a task, while products may, or may, not be produced continually during the task.

An extremely important issue facing plants is maintenance and cleaning. Models may have to deal with changeovers such as these. Since a unit may perform more than one task, the unit might require cleaning or recalibration from one task to the next. Scheduled maintenance may also require changeover. It is not uncommon for a unit to require preventative component change after a given number of batches produced.

For a comprehensive description of these particulars, the reader is referred to Méndez et al. (2006).

2.2 Discrete-Time Formulations

The discrete-time approach encompasses the discretisation of time horizons into homogeneous intervals. Critical points, such as the beginning and ending of tasks are confined to the boundaries of these intervals.

Although there had been much work done on discrete-time representations in the field of operations research (in the late 1950s and early 1960s), especially relating to the jobshop scheduling problem, it has only relatively recently (early 1990s) reached the domain of the generalised chemical engineering scheduling problem. Notable chronological developments in this regard were presented in Kondili et al. (1993), Shah et al. (1993), Pantelides (1994), Pekny and Zentner (1993), Zentner et al. (1994), Dedopoulos and Shah (1995), Bassett et al. (1996) and Elkamel et al. (1997).

The clear advantage of a discrete-time approach is its mapping of operations to equipment through time. This allows for a simpler formulation regarding constraints involved. The two seminal papers Kondili et al. (1993) and Shah et al. (1993) are used to illustrate this. Both papers make use of the STN representation (as shown in Figure 2.4). The assignment of units to tasks is a principal component to scheduling problems. Kondili et al. (1993) introduced this assignment through the use of binary variables, W_{ijt}, which decides whether a task i starts in unit j at time point t. Thus, allocation constraints are governed by:

$$\sum_{i \in I_j} W_{ijt} \leq 1, \qquad \forall j \in J, t \in T, \tag{2.1}$$

where J is the set containing all units and T, the set of all time points in the time horizon, I is the set of all tasks and I_j the set of all tasks which can be performed by unit j. This ensures that no more than one task can take place in a unit within a given time interval. To ensure that if a unit j is currently busy with task i, that no other task may begin in said unit until task i is finished, then the following constraint is required:

$$\sum_{i' \in I_j} \sum_{t'=t}^{t+\alpha_{ij}-1} W_{i'jt'} - 1 \leq M(1 - W_{ijt}), \qquad \forall j \in J, \forall t \in T, i \in I_j, \tag{2.2}$$

where α_{ij} is the associated processing time required for task i in unit j. Equation (2.2) is enforced only in the active case of $W_{ijt} = 1$. M is chosen as arbitrarily large enough to ensure that the left hand side of the equation is bounded less than equal to one.

Capacity limitations and material balances are also considered. The given amount of material (B_{ijt}) that undergoes task i in unit j at some time t is governed by the maximum and minimum capacities of the unit in question and are therefore constrained by:

$$W_{ijt}V_{ij}^{\min} \leq B_{ijt} \leq W_{ijt}V_{ij}^{\max}, \qquad \forall i \in I, t \in T, j \in J, \tag{2.3}$$

where $V_{ij}^{\max}$ and $V_{ij}^{\min}$ are the maximum and minimum capacity of unit j for task i. Batch sizes are forced to zero in the case of an inactive unit ($W_{ijt} = 0$). Additionally, intermediate materials in state s should not exceed maximum storage capacity for the state, C_s, given by:

$$0 \leq S_{st} \leq C_s, \qquad \forall s \in S, t \in T. \tag{2.4}$$

The material balances are handled by:

$$S_{st} = S_{s,t-1} + \sum_{i \in I_s^p} \rho_{is}^p \sum_{j \in J_i} B_{i,j,(t-\alpha_{is})} - \sum_{i \in I_s^c} \rho_{is}^c \sum_{j \in J_i} B_{i,j,t} + R_{st} - D_{st}, \quad \forall s \in S, t \in T, \tag{2.5}$$

where ρ_{is}^p and ρ_{is}^c are the proportions of state s produced and consumed by task i respectively. The sets of tasks that produce and consume state s are given by I_s^p and I_s^c respectively. Equation 2.5 implies that the increase or decrease of material in a given state s at time t is defined as the difference of the amount produced and consumed in the state. Initially Kondili et al. (1993) omitted variables R_{st} and D_{st}, however, they included this in various cases: (i) catering for the scenario where

customers expect a number of quantities, D_{st}, of material in state s at some time t; (ii) price fluctuations and/or lack of local storage leading to, R_{st}, quantities of raw materials being fed to state s at some time t.

Another important key in process scheduling problems is sequence dependant cleaning. This deals with the style and time required to clean between two tasks. Importantly, the identity and ordering of these tasks plays a major role in the level of cleaning. Kondili et al. (1993) illustrated this through the use of a dye manufacturing plant. The production of dark dye followed by a lighter dye clearly requires greater cleaning. The converse is not true however, where little to no cleaning may be required. Mathematically, these can be introduced with cleaning task $i_{jff'}$, describing the changeover needed for unit j to switch from family f to family f'. By family, we mean the family of tasks. For example, in the dye scenario above, we would have a *dark* family along with a *light* family. This cleaning task is then enforced if a task from family f is followed by family f' in the same unit j. The constraint is written as follows:

$$\sum_{t=t_1+1}^{t_2-1} W_{iff',j,t} \geq \sum_{i \in I_j^f} W_{ijt_1} + \sum_{i \in I_j^{f'}} W_{ijt_2} - \sum_{t=t_1+1}^{t_2-1} \sum_{i \in I_j} W_{ijt} - 1, \quad \forall j \in J, f, f' \in F, t_1 < t_2, \tag{2.6}$$

where I_j^f and $I_j^{f'}$ are the sets of tasks associated with family f and family f' respectively. This constraint is only imposing under one situation. That is when the first two summations of the right hand side each equal one, while the third summation assumes zero. A description of this is if item j begins a task of family f at some time t_1, and a task of family f' at time t_2 when no other task has been initiated between these two times. In the cases of extended time horizons, Equation (2.6) can lead to numerous constraints of approximately $\mathcal{O}(T^2)$. To counteract this, a separate constraint is utilised under the sufficient guarantee that if the unit starting family is f, it cannot begin family f' until the duration $\tau_{jff'}$ expires. This is the associated cleaning duration required for unit i to switch from family f to f'. The constraint

follows:

$$\sum_{i' \in I_j^{f'}} \sum_{t'=t+\alpha_{ij}}^{t+\alpha_{ij}+\tau_{iff'}-1} W_{i'jt'} \leq M(1-W_{ijt}), \qquad \forall j \in J, i \in I_j^f, t \in T. \tag{2.7}$$

Of course, there are numerous other relationships pertinent to specific scheduling problems. Examples include deliverables at intermediate due dates, renewable resources and importantly those mentioned in the South African context earlier. These additional constraints are implemented similarly to those listed above, and allow for solvable mathematical programming problems.

As mentioned before, the downside to these discrete-time based models is the potential infeasibility of enumeration in the mixed integer linear program (MILP), caused by discretisation. The process of approximating a continuous variable by a discrete one leads to error. It is no different here, with a major issue being the selection of uniform time intervals. Given the question of accuracy versus computational time, caution should be applied. There are, of course, cases which may be modelled accurately, such as processes which involve fixed processing times. These cases can be handled by discretising according to the HCF of all of the processes times. Even under these conditions, however, combinatorial issues arise, with likely only a small-time horizon solvable. Intractability increases when finer approximations are made to the discretisation. Further, the inclusion of processes with variable processing time allow for breakdown in bounded accuracy, due to their duration being of variable length. The tracking of this propagated error is also another difficulty, as the overall conditioning of the system may be sensitive.

As with all seminal work, extensions were done, specifically to counteract the difficulty of solving the cases of large MILP problems. These extensions came

through the exploitation of problem specifics. For example, the reformulation of the model where the gap between the MILP relaxed solution and optimal solution is reduced. Allocation and batch-sizing constraints were reformulated by Sahinidis and Grossmann (1991), Shah et al. (1993) and Yee and Shah (1998). The inclusion of cut constraints, which allow for a reduced search space, were implemented by Dedopoulos and Shah (1995) and Yee and Shah (1998). The pruning procedure of the branch and bound was improved by Shah et al. (1993) by reducing the size of the relaxed linear programme (LP). Divide and conquer strategies were also implemented, such as Bassett et al. (1996), with time-based decompositions and Elkamel et al. (1997) with both time and spatial decompositions.

2.3 Continuous-Time Formulations

The limitations of discrete-time formulations lead to the significant development of continuous-time formulations over the last twenty years. Continuous-time approaches are found in both sequential processes as well as the network represented processes listed in Figure 2.3. As mentioned in Section 2.1.2, the differences between these two are the omission of mass balance considerations in sequential processes due to their batch orientated nature. We shall discuss the two cases separately, firstly with sequential processes and then the general network represented process.

2.3.1 Sequential Processes

The earliest work for single and multiple stage sequential processes was done through use of time slots. In each stage, there may be one or several parallel units, as illustrated in Figure 2.7. Here time slots are associated with units and if multiple units are present, time slots are mapped to each individual units as seen in Figure 2.8.

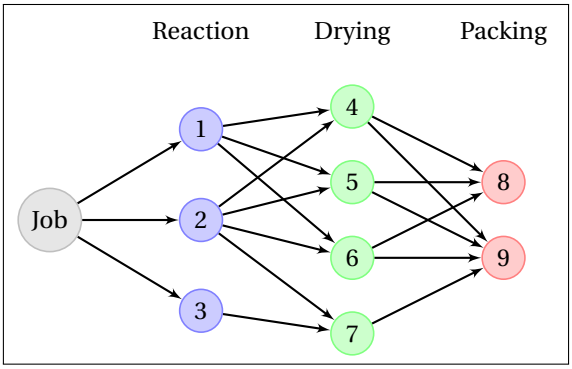

FIGURE 2.7: Example of Multiple Stage Sequential Structure (Adapted from Méndez et al. (2006))

Initial developments of sequential processes utilising this time slot approach were undertaken by Pinto and Grossmann (1994, 1995, 1996), Karimi and McDonald (1997), Pinto and Grossmann (1998), Bok and Park (1998), Moon and Hrymak (1999),

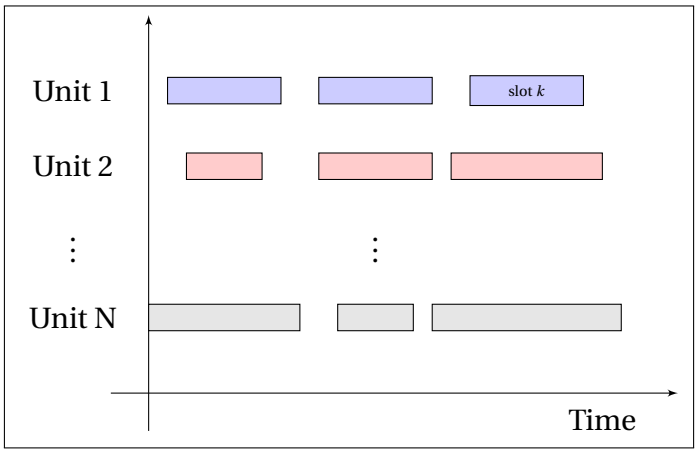

FIGURE 2.8: Example of Time Slots Assigned to Units (Adapted from Floudas and Lin (2004))

Chen et al. (2002), Lamba and Karimi (2002a, 2002b) and Lim and Karimi (2003).

A multistage flowshop problem containing parallel units at each stage is considered in Figure 2.7. The binary variable W_{ijkl} determines whether stage l for a given batch i is assigned to the time slot k for a given unit j. Continuous variables Tsi_{il} and Tei_{il} are the starting and end times of stage l for batch i respectively. Within stages, the continuous variable Ts_{jk} gives the starting time of time slot k for unit j. These allocations, as introduced by Pinto and Grossmann (1995), are constrained by:

$$\sum_{j\in(J_i\cap J_l)}\sum_{k\in K_j} W_{ijkl} = 1, \qquad \forall i\in I, l\in L_i, \tag{2.8}$$

$$\sum_{i\in I_j}\sum_{l\in(L_i\cap L_j)} W_{ijkl} \leq 1, \qquad \forall j\in J, k\in K_j. \tag{2.9}$$

To ensure that start times of batches match time slots as well as ensuring the start time of an order at some stage is correlated to the ending of the order at the previous stage, then:

$$-M(1-W_{ijkl}) \leq (Tsi_{il}-Ts_{jk}) \leq M(1-W_{ijkl}),\ \forall i\in I,\ J\in(J_i\cap J_l),\ k\in K_j,\ l\in L_i, \tag{2.10}$$

$$Tei_{il} \leq Tsi_{i,(l+1)}, \qquad \forall i\in I,\ l\in L_i-\{l_i^l\}. \tag{2.11}$$

This approach has two inherent problems. Firstly, the enumeration required is large due to the binary variable W_{ijkl} which exists over four dimensions. Furthermore, from an accuracy point of view, the number of time slots is defined beforehand ensuring a loss of guaranteed optimality.

Further extensions exploit the notion of batch order in sequential processes. Specifically these approaches take advantage by defining continuous variables to directly represent the timings of batches while omitting time slots. Works can be found in Moon et al. (1996), Cerdá et al. (1997), Méndez et al. (2000), Méndez et al. (2001), Hui et al. (2000), Orcun et al. (2001) and Lee et al. (2002). A comparison is drawn between a slot based model and a non-slot based model in the review by Floudas and Lin (2004). Specifically, they compare the non-slot based model proposed by Méndez et al. (2001) to the slot based model proposed in Pinto and Grossmann (1995). The comparison concludes that the resultant non-slot based model yields comparable complexity however it returns an improved objective value. The paper also reports a significant increase in CPU performance towards the non-slot based model, however, due to the fact that these times are achieved on different hardware, any claim on CPU performance improvement is unverified.

2.3.2 General Network-Represented Processes

In plants were batches are to be merged/split and have specific mass balance requirements, a general network-represented process can be utilised. Under continuous-time formulations, two main approaches are present: (i) *Global event based models*, where event or time points are used for tasks and units, meaning the event points are independent of specific units and (ii) *Unit specific event based models*, where event points dependencies occur only with units. With unit specific event models, event points for a particular unit may or may not be the same as another unit. A conceptual working example is illustrated with a global event model

seen in Figure 2.9, while the corresponding unit specific event model is seen in Figure 2.10.

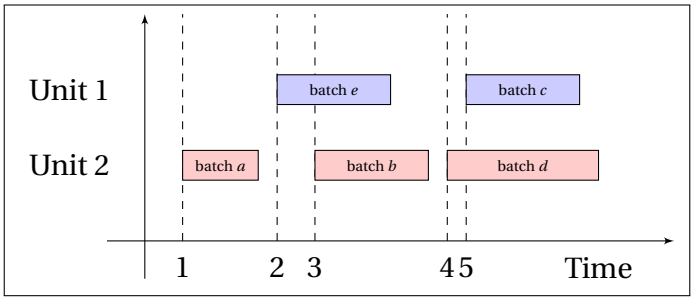

FIGURE 2.9: Example of global event points

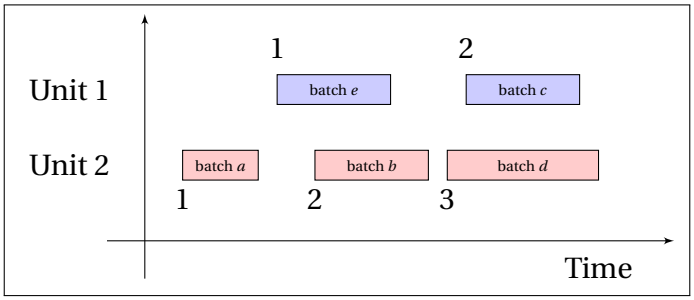

FIGURE 2.10: Example of unit specific event points

Global Event Based Formulations

Initial work on global event based formulation regarding continuous models was conducted by Zhang and Sargent (1996, 1998), Mockus and Reklaitis (1997), Mockus and Reklaitis (1999a, 1999b) and Schilling and Pantelides (1996, 1999). Notable extensions of these works were performed by Castro et al. (2002), Burkard et al. (2002) and Wang and Guignard (2002). Models found in these papers make use of either the STN or RTN representation, although the bias favours STN.

The premise of continuous-time models in this context is to allow continuous variables to determine the position of these event points and, at these points, map changes to the systems, such as the starting and ending of tasks, for example, Zhang (1995) and Zhang and Sargent (1996, 1998) implemented both STN and RTN for

combined batch and continuous process plants. The work by Zhang and Sargent (1996, 1998) lead to massive scale mixed integer nonlinear programmes (MINLP). This was due to linearisation (using techniques developed by Glover (1975)) of bilinear products of binary and continuous variables respectively. The cost of such a linearisation comes with the addition of vast numbers of variables and associated constraints.

The approaches found in Mockus and Reklaitis (1997, 1999a, 1999b) are based on the STN and are also implemented for an array of problems relating to multipurpose batch plants. The models themselves also lead to an MINLP. Exceptions to this are explored in cases when the overall objective function takes on a simpler form.

Schilling and Pantelides (1996, 1999b) approaches are continuous-time models based upon the RTN. Their work differs from Zhang and Sargent (1996), as well as Mockus and Reklaitis (1997) in the description of time slots and also the introduction of fresh binary variables. Similarly, the models lead to the requirement of linearisation for nonlinearity removal due to products of integer variables with that of continuous ones. To counteract the large enumeration required for the MILP (simpler cases) and MINLP problems, a intelligent modification of branch and bound was deployed, whereby the algorithm branches on both continuous time slots as well as the discrete variables.

Castro et al. (2002) contributed a RTN MILP formulation where binary variables denoted the start and ending of tasks at given event points. Continuous variables determined the position of the said event points. The RTN framework, from a constraint point of view, is similar to that developed in Schilling and Pantelides (1996).

As previously mentioned in Section 2.1.2, Majozi and Zhu (2001) introduced a new

process representation, the SSN framework. Here, the problem is developed into an MILP formulation. This approach treats event points as the use or production of some state. This is done through the introduction of the binary variable $y(s,p)$ which controls the usage of a given state s at a time point p. This approach leads to significant reduction in the number of binary variables. Care, however, must be taken when defining states since the states implicitly incorporate tasks and units.

Burkard et al. (2002) introduced an MILP model, utilising the STN framework, which considered the minimisation of the makespan for the batch processes as well as including further constraints. Wang and Guignard (2002) also propose a STN MILP model where events are linked to a change in inventory - thus reporting a reduction in required number of event points needed to produce a legitimate schedule.

When considering the global event based model, one can include a plethora of different factors in process scheduling. Many are explored in the greater body of literature, however, they are all affected by one major concern. This concern is the determination of the number of event points required to solve the model over a given time horizon. Most literature stated estimates, or adjusts, relative to specific problem size. This has been pointed out in the literature, specifically Zhang and Sargent (1996). Given a problem in which the number of event points are underestimated, the reality of suboptimal solutions is realised. Conversely, overestimation inflates feasible solution space and results in unnecessary additional computation, a situation to be avoided. Castro et al. (2002) attempted improvement in the matter, while iteratively increasing the number of event points in subsequent solutions. While the terminating condition is chosen to be a zero improvement from an additional event point, this is clearly not an effective metric as further additional event points may yield objective value improvement.

Unit Specific Event Based Formulations

Earliest work regarding unit specific event models was done by Ierapetritou and Floudas (1998a, 1998b), Ierapetritou et al. (1999), Lin and Floudas (2001) and Majozi and Zhu (2001). This novel approach established a new notion towards event points, where they represent sequential time points along the time horizon for each unit. These time points correspond to the initiation of tasks or the use of a unit. An illustration is found in Figure 2.10. These event points find themselves at different positions for different units. This allows for separate tasks to take place at different times in separate units for a given event point. This approach leads to a reduction in the number of event points when compared to that of its global event based model counterparts.

The introduction of binary variables which determine whether a task initiates at some event point, as well as whether a unit is active at an event point is presented in Ierapetritou and Floudas (1998a). These are controlled by $wv(i,n)$ determining if the task i begins at event point n, while $yv(j,n)$ determines whether unit j is active at event point n. To ensure that given some unit j at event point n, that only one task can take place in the unit, then the following is enforced:

$$\sum_{i \in I_j} wv(i,n) = yv(j,n), \qquad \forall j \in J, n \in N. \tag{2.12}$$

In the case of where plants have multiple units able to perform the same task, these tasks are then split into several tasks which will be each be performed in separate units. This does increase the set size of $wv(i,n)$, the worst case leading to the product of the set of all tasks with that of the set of all units.

Batches are determined by the continuous variable $B(i,j,n)$ giving the batch size of a given task i in unit j at some point n. The batch size is bounded through volume

constraints and the decision variable $wv(i,n)$ as such:

$$V_{ij}^{\min} wv(i,n) \leq B(i,j,n) \leq V_{ij}^{\max} wv(i,n),\ \forall i \in I,\ j \in J_i,\ n \in N. \tag{2.13}$$

The volume of a certain state, s, in the system is determined by the amounts stored $ST(s,n)$ and that delivered $D(s,n)$ respectively and is constrained by the material balance given by:

$$\begin{aligned} ST(s,n) &= ST(s,n-1) - D(s,n) + \sum_{i \in I_s^p} \rho_{is}^p \sum_{j \in J_i} B(i,j,n-1) \\ &- \sum_{i \in I_s^c} \rho_{is}^c \sum_{j \in J_i} B(i,j,n), \qquad s \in S, n \in N. \end{aligned} \tag{2.14}$$

The constraint given by Equation (2.14) is defined for batch tasks by Ierapetritou and Floudas (1998a), however, constraints regarding continuous tasks are discussed and derived in a follow-up paper by Ierapetritou and Floudas (1998b). The real valued times for the beginning and ending of tasks is measured through introduction of continuous variables $T^s(i,j,n)$ and the $T^f(i,j,n)$, the start and finishing times of task i in unit j at given event point n. In this particular case, the time required to complete a task is not simply just a fixed processing time but is also linearly dependant on the amount of material found in the unit to process. This is constrained by:

$$T^f(i,j,n) = T^s(i,j,n) + \alpha_{ij} wv(i,n) + \beta_{ij} B(i,j,n), \quad \forall i \in I, j \in J_i, n \in N. \tag{2.15}$$

A simplistic explanation of α_{ij} in Equation 2.15 is the time cost of turning unit j on to complete task i, while β_{ij} is the additional variable linear time cost for the same unit given some batch size B.

Importantly, there are also certain sequencing constraints that must be obeyed when considering these formulations. These manifest in: (i) the same task in the same

unit must follow at the end of the same task in the unit. Specifically, given task i to be initiated in unit j at an event point $n+1$, this must then begin after the unit has completed the same task from event point n. This is enforced by the constraint:

$$T^s(i,j,n+1) \geq T^f(i,j,n), \qquad \forall i \in I,\ j \in J_i,\ n \in \{N \setminus n_{\text{final}}\}. \tag{2.16}$$

Equation 2.16 holds for all event points except for the final event point in the set. Furthermore, (ii) separate tasks in the same unit are constrained by ensuring that task i initiated at event point $n+1$ begins post task i' at event point n given that the tasks both take place in unit j. This is implemented with:

$$\begin{aligned} T^s(i,j,n+1) \quad &\geq T^f(i',j,n) + \tau_{ji'i}\, wv(i',n) - H(1-wv(i',n)), \\ &\qquad j \in J,\ \{i,i'\} \in I_j,\ i \neq i',\ n \in N,\ \{N \setminus n_{\text{final}}\}, \end{aligned} \tag{2.17}$$

where H is the time horizon. Thus in such a case where $wv(i',n) = 1$ then Equation 2.17 implies that the starting time of task i in unit j at event point $n+1$ is forced to be the end time of task i' from event point n plus some additional cleaning required. Otherwise, the converse case is satisfied. Finally, (iii) in cases with different tasks in different units, the recipe order needs to be satisfied. The constraint follows:

$$\begin{aligned} T^s(i,j,n+1) \quad &\geq T^f(i',j',n) - H(1-wv(i',n)), \\ &\qquad \forall \{j,j'\} \in J,\ i \in I_j,\ i' \in I_{j'},\ i \neq i', n \in N,\ \{N \setminus n_{\text{final}}\}. \end{aligned} \tag{2.18}$$

Given that task i must be performed after task i' and should task i' take place in unit j' at event point n, then task i undertaken in unit j must start after the completion of task i' in unit j', i.e. $T^s(i,j,n+1) \geq T^f(i',j',n)$.

Various other nuances regarding waiting conditions between tasks or concerns such intermediate due dates can be included with additional constraints. Discussions

such as these can be found Floudas and Lin (2004, p. 2121).

Lin et al. (2004) introduced a short-term scheduling technique with an enhanced formulation of the model by Ierapetritou and Floudas (1998a). The enhancement allowed for tasks to continue over a number of consecutive eventpoints, thus allowing for improved accuracy in monitoring resources and storage. Their model was compared against the global eventpoint model of Maravelias and Grossmann (2003) with results showing that a notable reduction in the number of eventpoints was achievable. A comparative study between different continuous-time models for short-term scheduling was conducted by Shaik et al. (2006). The study dissected different models into the relevant subclasses of slot-based, global event based and unit specific event based formulations. Furthermore, the paper investigated the performance of these models when applied to several literature examples. This performance was measured under two objectives functions, namely the maximisation of profit and the minimisation of makespan.

As mentioned in the global event based models discussion, the concern of determining the number of event points is also present with unit specific based models. These are generally determined by applying an iterative procedure. Beginning with a small initial number of event points, this number is increased through each iteration terminating on the condition that the objective value no longer returns an improved solution. A point worth mentioning is that literature tends to report CPU time incorrectly because of this, by claiming the runtime of the final iteration only and not the total sum across all iterations. However, due to the better implementation of event points over global event based models, the probability of additional event points improving the overall objective value is smaller with unit specific based models.

The continuous-time models certainly lead to a smaller number of binary variables with regards to respective MILP models, which in turn results in shorter CPU times. Much literature is quick to point this out: certainly those implementations aiming to do so. However, many compare results in a manner not supporting claim. It appears almost always that CPU times are compared between models which have been implemented on completely different hardware. If CPU times are to be the metric of performance then all methods to be compared must be implemented on the same test system, i.e. hardware as well as software. This must also be verified over sufficient trials to ensure average times are obtained, at least on times short enough where the machine particulars may have influence.

Another metric used in the comparison of models is the number of binary variables between formulations. This, of course, is important if the formulations being compared are all MILP or MINLP models but has no real measure for an implementation not of this form. For a comparison of discrete-time vs continuous-time models, as well as a comparison between different continuous-time formulations, the reader is referred to Floudas and Lin (2004, Tables 2 & 3).

2.4 Additional Techniques for Batch Process

While both the discrete-time and continuous-time approaches are discussed in detail in Sections 2.2 and 2.3 respectively, there are a number of other approaches that have been developed for the scheduling of batch processes. These are subsequently discussed in brevity since the scope of this research is only to investigate frameworks involving the generalised discrete-time and continuous-time approaches.

2.4.1 Graphical Techniques

A novel graph representation for chemical processes in scheduling was introduced by Sanmarti et al. (1998, 2002). This work had the advantage of a considerable reduction in computational intensity. This reduction in computational complexity was due to no presence of binary variables since the time horizon was not required to be discretised. In this approach once all processing tasks are correctly represented in the recipe, optimal schedules are obtained utilising the S-graph framework. This S-graph was constructed with production tasks represented by nodes and precedence relationships via arcs. The longest path algorithm was introduced by Sanmarti et al. (1998, 2002) and allowed for the makespan of the schedule to be calculated.

The original work by Sanmarti et al. (1998, 2002) was limited to the minimisation of makespan (Majozi et al., 2017). Majozi and Friedler (2006) extended the capability of the S-graph to maximise throughput over a given time horizon. This extension was achieved through the introduction of a novel node-cutting algorithm. While this graphical approach may seem advantageous over mathematical programming techniques due to the simplified computational complexity, the actual generation of the S-graphs themselves are complex and sensitive to storage configurations (Majozi et al., 2017).

2.4.2 Precedence Sequencing

Precedence sequencing is another area that has seen active research. To minimise tardiness and earliness of orders Hui et al. (2000) introduced a continuous-time model based on immediate precedence sequencing. The approach by Hui et al. (2000) allowed for the considerable reduction of binary variables and as a result led to substantially reduced computational time. Importantly, this computational reduction came with no loss in the model's generality.

A continuous-time MILP framework for the scheduling of resource constrained multistage batch plants which supplied intermediates to end product facilities at predetermined time intervals was introduced by Méndez et al. (2000). This model was applied to a real-world two stage batch plant and the size of the model was shown to be considerable smaller than that of previous approaches. Méndez et al. (2001) introduced a new formulation for multiproduct batch plants with restricted supplies of discrete resources. This model was compared against the model of Pinto and Grossmann (1995) and achieved improved objective values at reduced computational times.

Medium-term production for multiproduct scheduling was considered by Lin et al. (2002). This approach involved first decomposing the time horizon into shorter successive short-time horizons and then iteratively applying a continuous-time formulation for short-term scheduling until the entire time horizon was completed.

Gupta and Karimi (2003b) introduce a two-step MILP based upon immediate precedence sequencing for non-continuous multistage plants. An improved MILP formulation for the scheduling of multiproduct multistage plants was introduced by Gupta and Karimi (2003a).

A general precedence sequencing approach for the scheduling of multistage multiproduct batch plants was introduced by Ferrer-Nadal et al. (2008). A key contribution by Ferrer-Nadal et al. (2008) was that non-zero transfer times were considered. This is contrary to majority of the literature which makes use of the

simpler assumption of zero transfer times. Their results showed that danger of making the zero transfer time assumption, especially in the cases where shared units and storage tanks are involved as this could severely affect the feasibility of the schedule.

Chapter 3

Metaheuristic Approach

When considering the previous Chapter, one of the main concerns shared by all literature contributions is that of the increased solution times with scale on batch plants. Notwithstanding the eloquence of the model, applying mathematical programming to these problems will always lead to this intractable enumeration. Within the literature, often at medium-to-long time horizons these models have not been solved to optimality and will have generally reached a cut-off time that has been specified by the solver used. The implication of this is that these models are often not solved to global optimality, but rather to a local optimum within some acceptable relative gap. Therefore, at termination of these computational runs only a close-to-optimal solution can be claimed. Importantly, these close-to-optimal solutions are often correspondingly reported with substantial CPU times.

If close-to-optimal solutions are acceptable at these medium-to-long time horizons, then mathematical programming need not necessarily be the approach to consider. Metaheuristics have often been used when intractable problems are considered and with great effect, either finding global or close-to-optimal solutions. The research presented here aims to apply these metaheuristic ideas to the scheduling of batch plants. Prior work has been conducted in the application of metaheuristic techniques to batch processes, such as a genetic algorithm (GA) by Azzaro-Pantel et al. (1998) showing promise for large scale combinatorial problems. Computational

speed between mixed integer linear programmes (MILP) and a genetic algorithm in the scheduling of chemical batch plants was investigated by Cantón Padilla (2003). However, the majority of this early metaheuristic work either focussed on the design of batch plants using metaheuristics such as Cavin et al. (2004) or the single-stage or multiproduct batch plant scheduling as found in He and Hui (2007, 2006). In terms of solving multipurpose process scheduling through metaheuristics, very little literature exists. An extremely fast GA which comprehensively outperforms top performing MILP models by orders of magnitude in computational time was introduced by He and Hui (2010). Unfortunately, the framework in which the GA is designed exploits the nature of the case study solved. Although this is a useful contribution, it lacks a generalised framework to apply to other problems.

As discussed in Chapter 2, approaches to batch scheduling commonly belong to either a discrete-time or continuous-time formulation. Therefore, a metaheuristic approach is developed for both of these time formulations.

The remainder of this Chapter is as follows. In Section 3.1 the specific metaheuristic techniques utilised are discussed. Section 3.2 outlines the problem description with Sections 3.2.1 and 3.2.4 introducing the discrete-time and continuous-time approaches respectively.

3.1 Metaheuristics Utilised

Three metaheuristics are discussed and applied to the two literature examples considered. All three are nature-inspired algorithms, two of which are well known and established, while the third is a relatively new method. The algorithms are briefly discussed in the Sections 3.1.1 through 3.1.3, with the details of each algorithm elaborated further in Sections 3.2.1 and 3.2.4 respectively.

3.1.1 Genetic Algorithm

The **Genetic Algorithm (GA)** is an optimisation algorithm commonly used to find optimal solutions to computational problems. This method was introduced as early as 1962 by John Holland in his work on adaptive systems with the seminal work published by Holland (1975). Genetic Algorithms fall under the evolutionary algorithms branch of metaheuristics as they make use of the biological processes of natural selection and species reproduction in order to determine the fittest solutions to specific problems. The algorithm takes as input a set of potential solutions (chromosomes) to a problem encoded in a particularly relevant manner. Generally, this input is accompanied by a metric or *fitness function* used to measure the potential of each candidate solution. These input candidate solutions may already be known or belong to some initial feasible set. However, more often than not these candidates are randomly generated. The algorithm then attempts to improve these solutions through its invocation.

The GA works by evaluating each solution with the required fitness function and ranks them accordingly. Often, with solutions being randomly generated, many solutions are either poor or infeasible and will be bred out of the population. However, there may be some of these candidate solutions which have some potential in solving the problem. These candidate solutions are kept within the population and allowed to reproduce and produce progeny. These progeny carry the genetic make-up of their parents with random changes introduced through inexact copying, similar to that of real species reproduction. These progeny move onto the following generation forming a new population of candidate solutions. The algorithm then repeats another round of fitness evaluation. Poor or weak candidates are again removed from the population, while hopefully, the random changes introduced into the population lead to some improved candidates, yielding a strong or more efficient set of populations applicable to the current problem. This process repeats

generation after generation with the expectation that the average fitness of the population increases over time, often with extremely accurate solutions to the problem.

There are numerous ways to decide on the encoding for genetic algorithms. Commonly this involves solutions being encoded as binary strings, i.e. sequences of ones and zeros, where the position represents some parameter value of the chromosome (Fulcher, 2008). Another popular approach is encoding the chromosomes as arrays where entries may be integer or decimal values, again where each bit represents a specific feature of the chromosome (Fulcher, 2008). It is this approach which will be applied here with further detail discussed in Sections 3.2.1 and 3.2.4.

With respect to the selection process of the genetic algorithm, many techniques have been used. Some examples include:

- **Elitism selection:** this strategy selects the fittest candidates at each generation, normally either the absolute best or some percentage of the best. These chromosomes are automatically copied from one generation to the next.
- **Roulette-wheel selection:** this strategy selects chromosomes based on chance. Each individual is assigned a probability of being selected based on amount by which its fitness value is superior or inferior to that of its competitors fitness.
- **Tournament selection:** this strategy selects subsets of chromosomes from the population and these subsets then compete against one another. The winners of each subset are chosen to reproduce.
- **Hierarchical selection:** this strategy has chromosomes pass through multiple rounds of selection at each generation. More rapid low-level fitness tests

are applied, with survivors then subjected to more rigorous evaluation. This is often used to speed up overall computational time as the faster low-level tests can quickly remove chromosomes which have little to no value and therefore only selecting more promising chromosomes to more computational expensive evaluation.

After the selection process, chromosomes are selected randomly to attempt to improve their fitness for the following generation. The two main strategies to achieve this are *crossover* and *mutation*. Crossover selects two chromosomes, called parents, and swaps selections of their solution producing two progeny. This processes of the algorithm is analogous to sexual reproduction. Common practice is single-point crossover where a point along the chromosome is randomly selected and the genome of the first parent up to this point is combined with the genome after this point from the second parent. This process is illustrated in Figure 3.1.

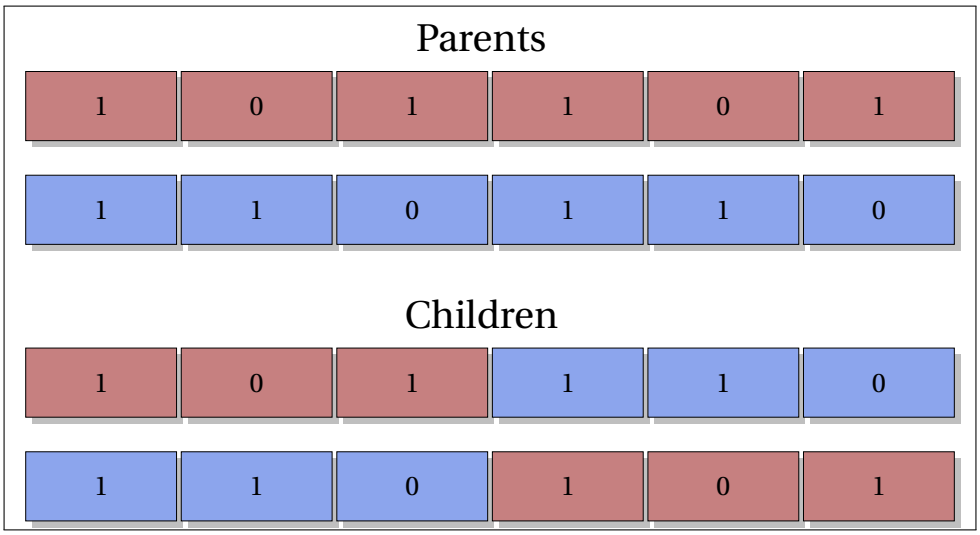

FIGURE 3.1: An Example of Crossover in Genetic Algorithms

The second method, mutation, is simple. As seen with mutation in species, genes are often changed from one to another. Thus, the genetic algorithm similarly causes a small change or alteration to a single point in a candidate chromosome. In the case of binary chromosomes this is called a *bit-flip*, where a bit is changed from either a zero to one or vice-versa. An example of this is illustrated in Figure 3.2.

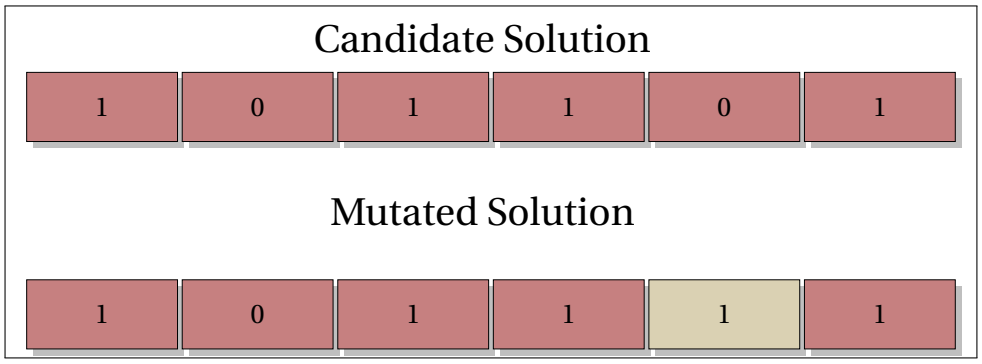

FIGURE 3.2: An Example of Mutation in Genetic Algorithms

Application of the algorithm is relatively simple and is illustrated in Figure 3.3. Generally, the parameters for this algorithm, i.e. the optimal rates of crossover (θ) and mutation (ψ) are unknown and need to be found under experimentation. However, conceptual methodologies for tuning these parameters do exist, with an excellent survey of these techniques found by Eiben and Smit (2011).

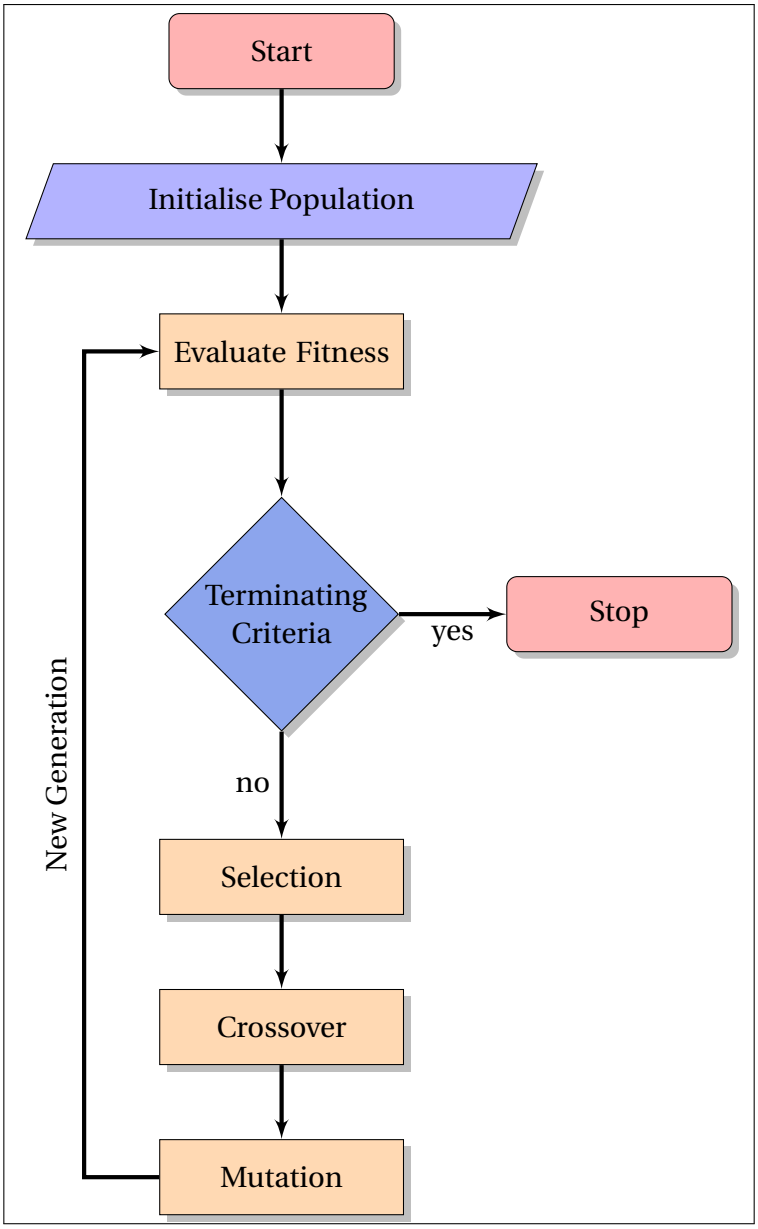

FIGURE 3.3: Flowsheet of the Genetic Algorithm

3.1.2 Simulated Annealing

Another naturally inspired metaheuristic is **Simulated Annealing (SA)**. This algorithm was introduced by Kirkpatrick et al. (1983) and is based on the physics problem of physical annealing. Kirkpatrick et al. (1983) showed that the model used for simulating the annealing process proposed by Metropolis et al. (1953) can also be used in computing the solutions to certain optimisation problems. The annealing process involves the cooling of materials which allow for recrystallization and a return of the material to its equilibrium. In contrast, the simulated annealing process utilises the fitness function of the problem instead of the energy of some material. Simulated annealing has been extensively applied to large scale optimisation, especially problems where a global optimum is hidden amongst many poorer local extrema. The algorithm is well proven, and for all practical points of view, has successfully solved the **Travelling Salesman Problem (TSP)**, along with many other problems as seen in Press (2007).

The algorithm itself is quite simple and is essentially a hill-climbing technique. However, a random move may be selected instead of the best move. Should a newly generated solution improve the current solution, then it is accepted. Alternatively, should the newly computed solution not improve the current solution, then the algorithm may make this move anyway - dependant on some computed probability. This probability will decrease exponentially depending on how poor the move may be. This worsening of the solution is analogous to an energy increase. This probability is based on the Boltzmann probability distribution:

$$\text{Prob}(E) \sim \exp\left(\frac{-E}{kTM}\right), \tag{3.1}$$

where E is the energy state, TM is the temperature of the system in thermal equilibrium and k is the Boltzmann constant. The Boltzmann probability

distribution describes the idea that a system, even at low temperatures might possibly be in a high energy state. In terms of an optimisation description, this means that a system will sometimes go uphill in addition to downhill, however, the lower the temperature the less likely this uphill execution will take place. The modification done by Metropolis et al. (1953) was as follows. Given a simulated system, this system will change its configuration from energy E_1 to energy E_2 with the probability:

$$p = \exp\left(\frac{-(E_2 - E_1)}{\nu TM}\right). \tag{3.2}$$

Therefore, in cases where $E_2 < E_1$ a value greater than one is realised and as a result Equation (3.2) is set to $p = 1$, meaning that the system will always take this downhill step. Conversely, $E_2 > E_1$ implies that sometimes an uphill step will also take place. The control parameter TM, in essence the “temperature", is cooled with the annealing schedule ν, which determines the rate at which high TM values are lowered. As with the genetic algorithm, these values often require either some physical insight to the problem or are attained through experimentation. The flowsheet of the simulated annealing algorithm is illustrated in Figure 3.4.

3.1.3 Migrating Bird Optimisation

Finally, the last metaheuristic applied is **Migrating Bird Optimisation (MBO)** which was recently proposed by Duman et al. (2012) to solve the **Quadratic Assignment Problem (QAP)**. Inspired by the flying “V" formation observed in flocks of migrating birds, it has been shown to be effective at tackling large scale problems by Meng et al. (2018) and other well known intractable optimisation problems such as the **Knapsack Problem (KP)** as done by Ulker and Tongur (2017). The primary reason birds utilise this formation is for energy saving. In fact, a study done by Lissaman and Shollenberger (1970) showed cases where flocks of birds achieved up to 71% increased flight ranges than that of a single bird. It is this energy minimisation that

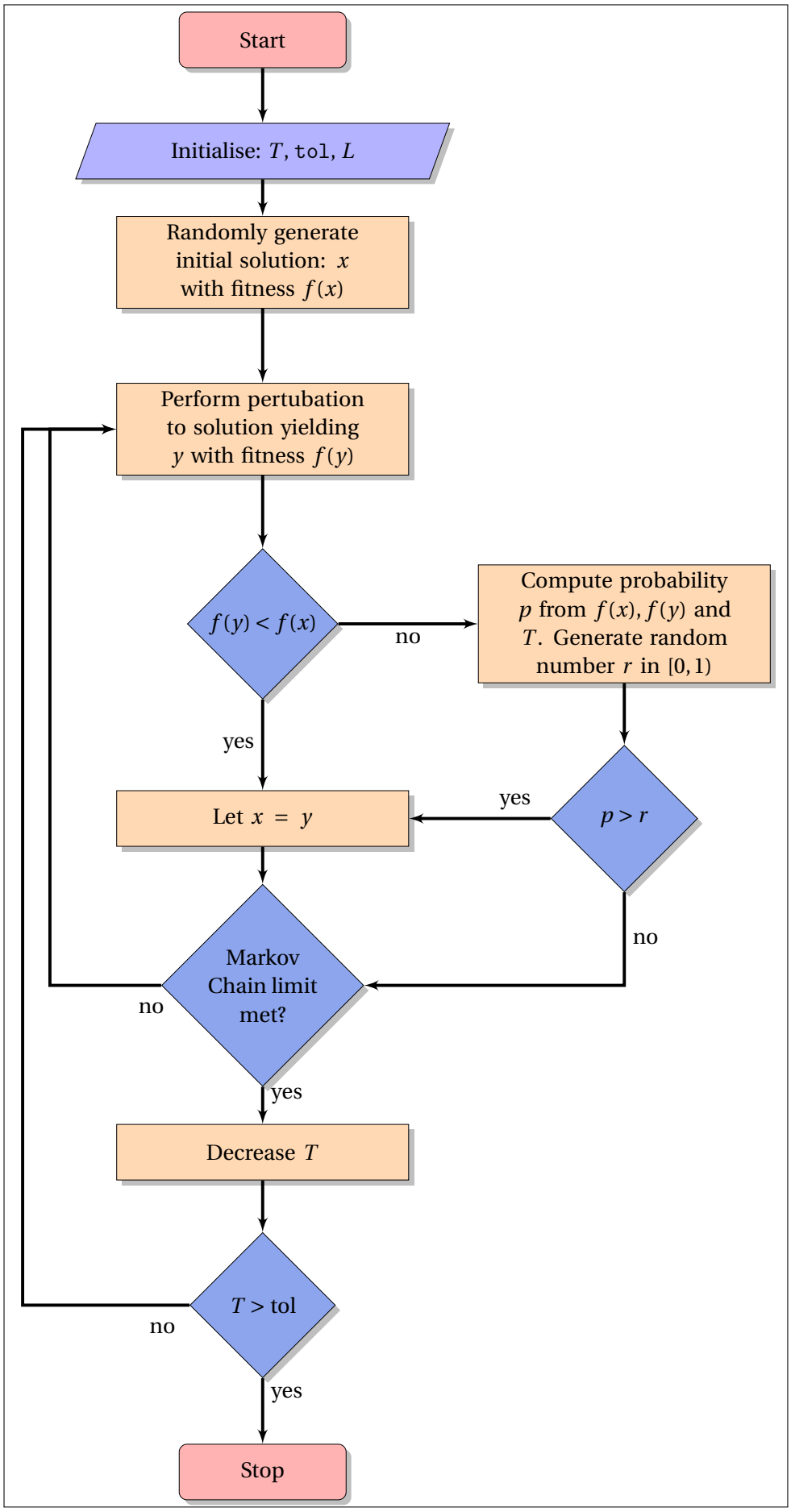

FIGURE 3.4: Flowsheet of the Simulated Annealing Algorithm

the algorithm seeks to mimic.

Within this formulation, a single bird leads the flock, which trail on both the left and right hand side of the leader respectively. As a result, all birds excluding the leader receive benefit caused via the up-draft generated by the number of birds preceding it. Over time, the leading bird tires and retreats to the tail of the flock for rest resulting in a new bird taking place of the leader. The MBO algorithm employs a neighbourhood search technique where the trial solutions are the birds flying in V formation. Beginning with the initial solution, i.e. the leader bird and continuing through the left and right tails, each solution is attempted to be improved upon via its neighbouring solutions. Should a neighbouring solution improve a current solution, then this solution is replaced by its neighbour. Importantly, the method invokes a benefit strategy, whereby trailing birds do not only consider candidate solutions from its own neighbourhood but also a subset of those solutions not used in the preceding birds' neighbourhood. Effectively, a bird considers its own neighbours along with the next best from the previous solution. After all solutions have been attempted to be improved via neighbouring solutions, the process repeats for a given number of times, known as tours, at which the first (or leader) solution becomes the final solution and one of the second level solutions becomes the first and another set of tours begins. The algorithm terminates after a pre-specified number of iterations.

The MBO algorithm is controlled through five parameters, q, the number of birds considered in the initial population, κ, the number of neighbouring solutions to consider, ξ, the number of neighbouring solutions to be shared and passed onto the next solution, η, the number of tours and K, the total iteration limit. The flowsheet of the algorithm is described in Figure 3.5.

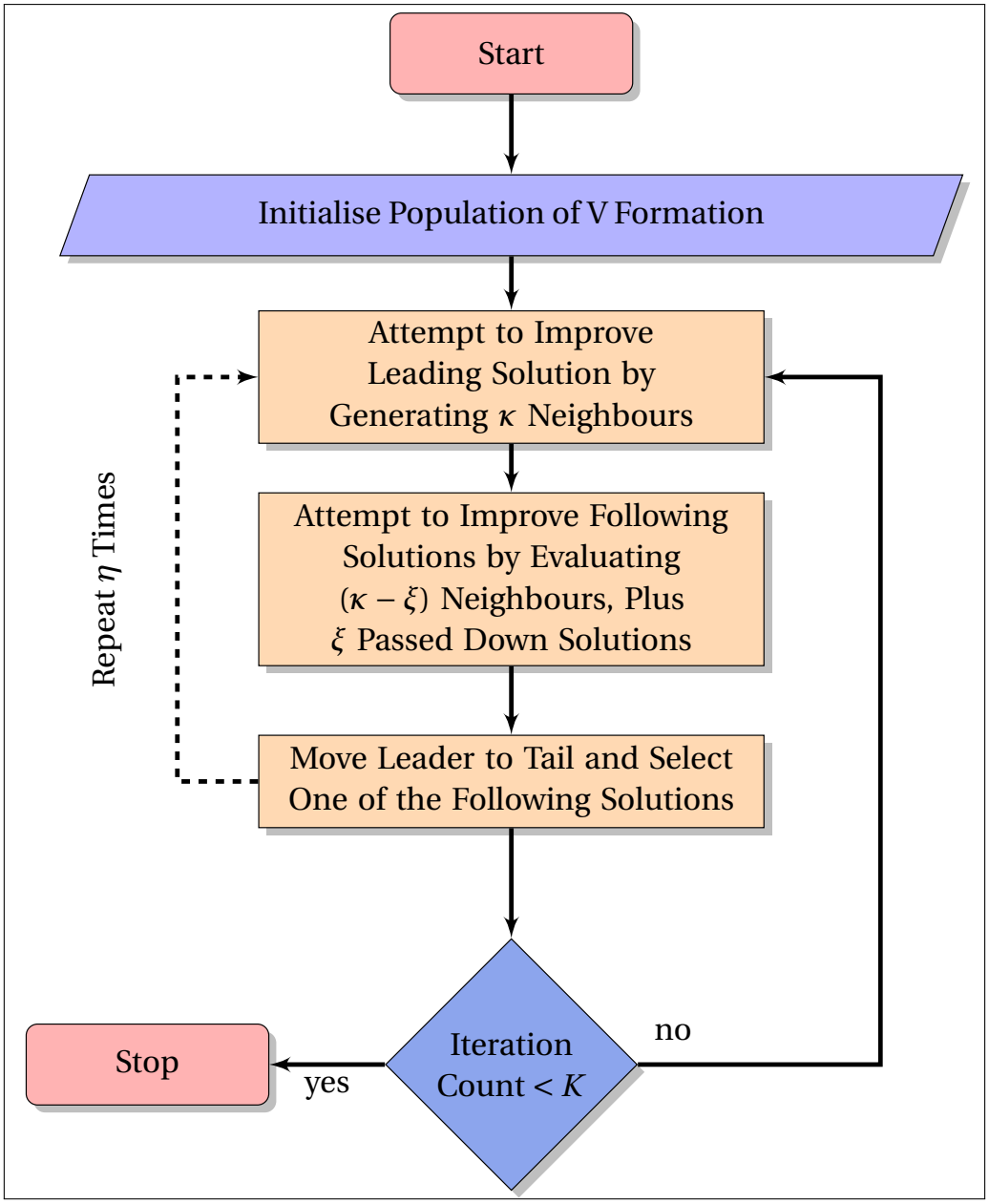

FIGURE 3.5: Flowsheet of the Migrating Bird Optimisation Algorithm

3.2 Problem Description

The metaheuristics outlined in Section 3.1 are applied to two literature examples commonly found in batch process scheduling literature. Firstly, we introduce the motivating example, introduced by Ierapetritou and Floudas (1998a) and then the primary example introduced by Kondili et al. (1993). As was discussed in Chapter 2, both discrete-time along with continuous-time formulations have been developed. Therefore, a approach to both of these scenarios is introduced utilising the aforementioned metaheuristics.

The motivating example is a simple multiproduct plant with three process, *mixing*, *reaction* and *purification*, with one final product produced. This example has been used frequently when introducing novel frameworks to batch scheduling. The plant flowsheet can be seen in Figure 3.6.

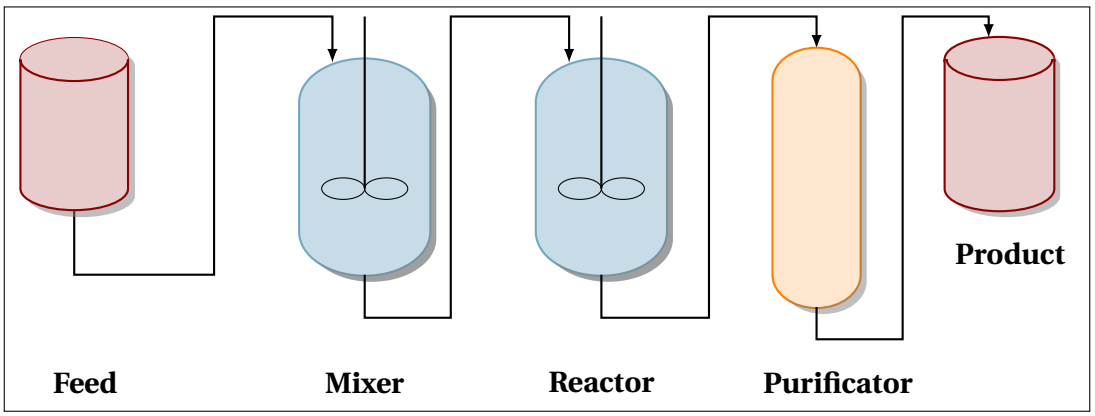

FIGURE 3.6: Flowsheet for the Motivating Example.

The primary example was originally introduced in Kondili et al. (1993) and has seen significant attention and use when considering a fresh formulation. The flowsheet for the primary example is illustrated in Figure 3.7. The problem consists of three raw feeds, two reactors - which are both capable of performing three different reactions - and a still performing separation. The plant produces two end products, both valued

at 10 Monetary Units (MU). The STN representation seen in Figure 2.4 illustrates the specific steps and sequences the raw material must follow to produce these products.

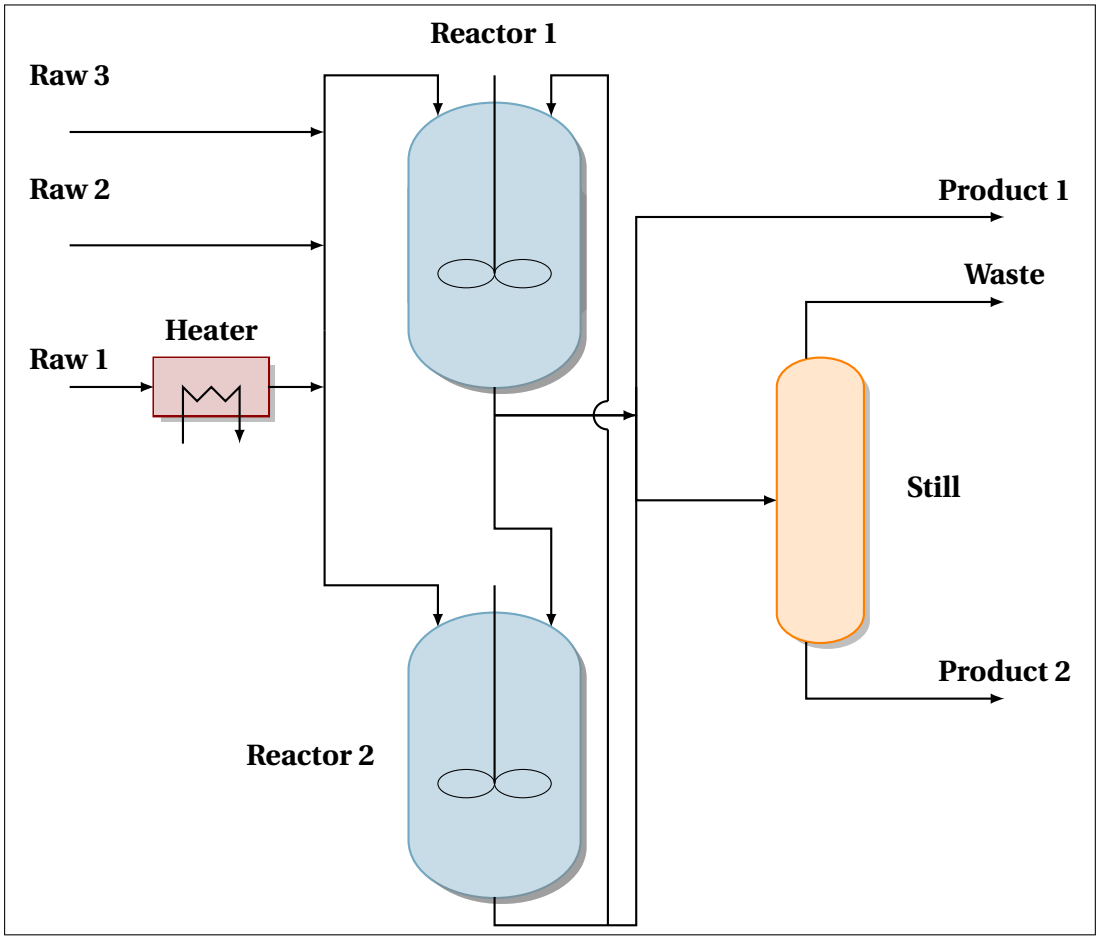

FIGURE 3.7: Flowsheet for the Primary Example.

3.2.1 Discrete-Time Approach

The first and easier approach is one following a discrete-time formulation. Here tasks are assumed to have fixed processing times, i.e. processing a batch in a unit has a fixed time regardless of how much material is processed. This assumption is made in order to easily define what granularity would be needed. In the examples considered, this granularity will be the highest common factor (HCF) of all tasks. For example, if this happens to be half-an-hour then a time point needs to be dedicated to every half-hour on the time horizon.

One of the major drawbacks for this approach was the introduction of a substantial amount of binary variables. While this has drastic implications for MILP formulations and solvers, it is substantially less of an issue for a metaheuristic approach as the operation complexity does not increase, rather it only increases the length of vector operations to perform, a task which modern computing equipment can handle with ease.

This idea is now introduced to both literature examples to formulate a representation that is feasible and intuitive for all the metaheuristics applied. The introduced strategy is not unique to these two cases and could be implemented on any other plant configuration.

3.2.2 Motivating Example

In order to utilise the metaheuristics, a consistent formulation is required which should describe the operation of any unit at any discrete point in time. An operations matrix, denoted i_m, can be used to achieve this. Here the number or rows represents the number of units and the columns represent time intervals. These intervals will be the HCF of all the units. To illustrate, consider the motivating example from Figure 3.6. The data for this example as introduced by Ierapetritou and Floudas (1998a) is given in Table 3.1. Here the three units have 4.5 hours, 3 hours and 1.5 hours processing times respectively. Therefore, in terms of the matrix representation we have a matrix of size $(3 \times \gamma H)$, where $\gamma \in \mathbb{N}$ is the number of time periods per hour and H the time horizon. To compute the number of time periods, γ, the HCF of the processing times is considered. In this particular case, the HCF is 1.5 hours. Therefore, two time periods per hour are required (i.e. $\gamma = 2$), since the tightest discretisation is one half-hour. The subsequent task is to determine the values of the elements within this operation matrix. Since each unit can only perform one task each element is binary, where one indicates the unit is active and zero indicates

TABLE 3.1: Problem Data for the Motivating Example with Fixed Processing Time.

Unit	**Capacity**	**Suitability**	**Processing time** (τ)
Unit 1	100	Mixing	4.5
Unit 2	75	Reaction	3.0
Unit 3	50	Purification	1.5
State	**Storage Capacity**	**Initial Amount**	**Price**
State 1	Unlimited	Unlimited	0.0
State 2	100	0.0	0.0
State 3	100	0.0	0.0
State 4	Unlimited	0.0	1.0

that the unit is inactive. A random binary operation matrix over an arbitrary time horizon is illustrated in Figure 3.8.

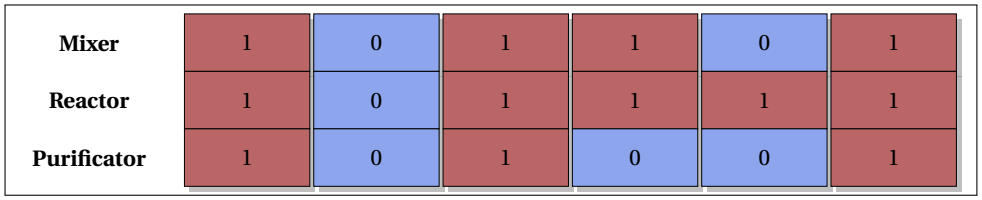

FIGURE 3.8: An Example of an Operations Matrix. Red implies Unit is Active and Blue implies Unit is Inactive

These operation matrices can be generated randomly and may or may not be feasible. That is not to say, however, that an infeasible matrix does not have any measurable fitness. It is the job of the fitness function to determine the possible feasible output from the matrix, since the objective is to maximise the volume of product produced. For example if a unit is currently active and will be for the next four time intervals, then any suggestion from the matrix to turn the unit on or off will be ignored until the operation is complete and will only then comply with the next feasible instruction.

This representation of instruction is compliant with all three metaheuristics. The different algorithms explore these solutions in their respective means. The fitness function applied is the same for all three techniques and obeys the same

constraints as outlined in Ierapetritou and Floudas (1998a). This fixed processing time configuration of the problem has been studied before. In particular, a comprehensive study with long-time horizon tests performed by Seid and Majozi (2012a).

Application of the Genetic Algorithm

The outline of the application of the GA to the motivating example follows:

1. The population is initialised through the generation of trial candidate solutions. This is done by generating the desired population size, P, where each member of the population is a non-uniform binary operations matrix as described above.

2. The algorithm is run for a set number of generations. At each generation, elitism is applied and selection of solutions to automatically survive to the subsequent generation will be the crossover rate multiplied by the population size, i.e. $\lambda = \theta \times P$, where λ is the elite population. This elitism implies that $(P - \theta P)$ progeny need to be generated in order to pass a full population to the next generation. This progeny is generated through the crossover operations. Parents are randomly selected from the surviving elite candidate solutions to generate the complement of the population. This operation utilises single-point crossover where a progeny solution is the concatenation of two parents at a crossover point as illustrated in Figure 3.9. This crossover point is defined by:

$$c_p = \{2, ..., \gamma H - 1\}, \tag{3.3}$$

   implying crossover may take place anywhere except for the endpoints of the operations matrix.

3. After the population has been replenished via the addition of progeny, a subset of these new solutions are subjected to the mutation operation, the number

of which is computed using the mutation parameter, i.e. $\psi \times (P - \lambda)$. The operation works by randomly selecting a bit in the operations matrix and performing a bit-flip as was illustrated in Figure 3.2.

4. Finally, this new population is ready for fitness evaluation at the next generation and the process repeats until the number of generations is reached.

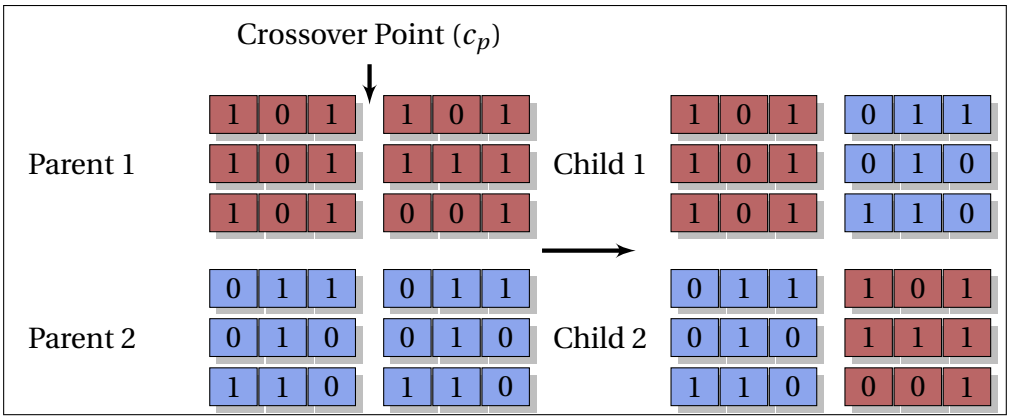

FIGURE 3.9: An Example of a Crossover Operation in term of the Motivating Example

Application of the Simulated Algorithm

The outline of the application of SA to the motivating example follows:

1. A number of initial solutions is generated and evaluated, with best operations matrix selected for the SA process.

2. The algorithm runs until the given starting temperature, TM is below some threshold value, ϵ, where the temperature is cooled at a rate of vTM and $v \in [0.9, 1)$. At each temperature, a number (LM) of perturbations are performed. The operation to apply for a perturbation is randomly selected from a number of different available operations. These operations include *swapping*, *inversion*, *translation* and *regeneration*. These operations are defined below:

- **Swapping:** the swapping operation randomly selects two different bits and swaps their values.
- **Inversion:** with inversion, a portion of the operations matrix is selected, specifically a subset of columns and all rows and its order inverted. This is illustrated in Figure 3.10.
- **Translation:** translation operations translate a selection of columns from operation matrix in some particular direction, as seen in Figure 3.11.
- **Regeneration:** regeneration randomly selects a column of the operations matrix and generates fresh random values for all bits in the respective column.

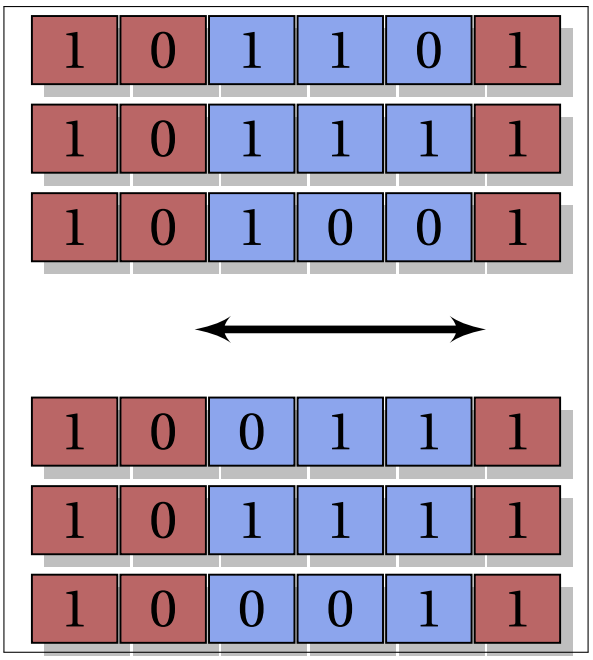

FIGURE 3.10: An Example of a Inversion Operation in term of the Motivating Example

Application of the Migrating Bird Optimisation

The outline of the application of MBO to the motivating example follows:

1. The population is initialised through the generation of trial candidate solutions. Extensive parameter tuning was undertaken by Duman et al. (2012)

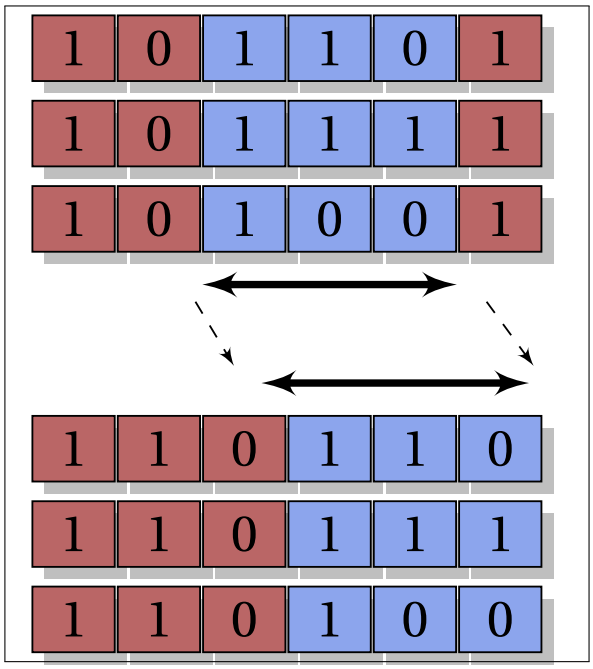

FIGURE 3.11: An Example of a Translation Operation in term of the Motivating Example

when the MBO was originally introduced. The MBO implemented for both the motivating and primary example utilises these optimal parameter settings. Specifically, the population size was fixed at $P = 51$, the number of tours each leader was required to perform was set at $\eta = 10$, the number of neighbouring solutions to generate was set at $\kappa = 3$ and the number of solutions to be shared with the next solution was $\eta = 1$. The maximum number of iterations was different and was set to $K = 100$ as this appeared to be sufficient under coarse testing.

2. The algorithm made use of three operations to generate neighbourhood solutions, namely, *bit-flip*, *swapping* and *regeneration* defined as:

   - **Bit-flip:** as in the genetic algorithm, a random bit is selected and its value flipped.
   - **Swapping:** as in the simulated annealing, a pair of random columns are selected and swapped positionally.

- **Regeneration:** as with the simulated annealing, a column is randomly selected and generated new random values.

3.2.3 Primary Example

The primary example is quite similar to that of the motivating example except that in this case some of the units can perform more than one process. This requires very little change to the representation. On units where more than one process is available the integer range of the bit is changed from binary to the set length of processes available. Considering the problem data given in Table 3.2, two of the four units are able to perform multiple processes, three in totality. The remaining two are still binary, since the heater and still are either on or off. As a result, the change requires only the rows corresponding to reactor 1 and 2 to be allowed to sample from the integer range of [0,3], with 0 still implying off and 1 - 3 implying reactions one through three respectively.

TABLE 3.2: Problem Data for the Primary Example with Fixed Processing Time.

Unit	**Capacity**	**Suitability**	**Processing time** (τ)
Heater	100	Mixing	1.0
Reactor 1	50	Reaction 1, 2, 3	2.0, 2.0, 1.0
Reactor 2	80	Reaction 1, 2, 3	2.0, 2.0, 1.0
Still	200	Separation	1.0 for product 2, 2.0 for IntAB
State	**Storage Capacity**	**Initial Amount**	**Price**
Feed A	Unlimited	Unlimited	0.0
Feed B	Unlimited	Unlimited	0.0
Feed C	Unlimited	Unlimited	0.0
Hot A	100	0.0	0.0
Int AB	200	0.0	0.0
Int BC	150	0.0	0.0
Impure E	200	0.0	0.0
Product 1	Unlimited	0.0	10.0
Product 2	Unlimited	0.0	10.0

One of the major advantages of this representation is that it allows the plants to be vastly more complicated without the need for generating longer candidate solutions. Since the goal is to produce maximum product, the length of the solution is solely governed by the time horizon only. Adding complexity such as further processes or stricter constraints does not affect the solution representation as this is the responsibility of the fitness function. Furthermore, while this increased complexity may add to the fitness function, the representation still allows for the function to be evaluated cheaply as once an operation is active the fitness function need not access any suggestions for an active unit until its current task is complete.

Application of the Metaheuristics

Importantly, nothing intrinsically changes with the application of the metaheuristics to the primary example. However, a new fitness function is required and follows the constraints introduced by Kondili et al. (1993), which is somewhat more complicated than that of the motivating example. Similarly, the algorithms perform in the same fashion as the motivating example; the only differences being the change required when perturbations are applied on units which are not binary. As a result, bit-flips in this regard are not flipped per sé, but resampled from the integer set of operations available. Due to the geometry of the representation in its matrix form, column swaps, regenerations, translations and inversions do not need to be changed in any way. This allows for extremely robust algorithms as very little change is required when moving from one plant configuration to the next.

3.2.4 Continuous-Time Approach

The second major framework introduced applies a continuous-time approach to the solution of the two literature examples. In this framework, tasks completed in units are assumed to have variable processing times, i.e. the time to processes a batch is dependant on the amount of material the unit is required to process.

This assumption has a significant impact on the complexity of the problem and requires the introduction of an additional candidate solution matrix. Specifically, the framework makes use of a coupled system, where a candidate solution posses both an instruction matrix deciding the operations to perform, and a time interval vector deciding on the durations of these respective tasks.

The operations matrix is described in the same manner as in the discrete-time approach. However, the length (number of columns) of this matrix is subjected to change. Instead of having a fixed number of columns as determined by the time horizon and the HCF of all units, the number of columns is now the given number of eventpoints. This continuous-time approach is analogous to global eventpoint approaches developed in the MILP literature, whereby number of eventpoints, n, is not unique to individual units. Rather, these eventpoints describe an interval of time in which a set of instructions are to be performed. Therefore, the size of the instruction matrix will always have the number of all units × the number of eventpoints, i.e. $J \times n$.

The time interval vector represents the durations of a corresponding instruction matrix. That is, the vector represents the durations for which sets of instructions should run. The time interval candidate will have size, one × the number of eventpoints, i.e. $1 \times n$ where each element of the vector is the duration for which the corresponding column in the instruction matrix is allowed to perform. For example, if 4 eventpoints are selected for a time horizon of 12 hours, then four arbitrary values are generated which sum to the length of the time horizon, i.e:

$$\sum t_v(H,n) = \sum t_v(12,4) = \sum [4.665 \quad 3.479 \quad 2.427 \quad 1.429] = 12, \tag{3.4}$$

where t_v is the time interval vector. Here the indexed values correspond to the same instruction index (column) of the opposing instruction matrix i_m. A complete

TABLE 3.3: Motivating Example optimal solution in proposed framework.

t_{v_i}	4.665	3.479	2.427	1.429
Mixer	1	1	1	1
Reactor	0	1	1	1
Purifier	0	0	1	1

example of this coupled framework is seen in Figure 3.3, which coincidently is the optimal solution to the motivating example over the 12 hour time horizon. Elaborating on Table 3.3 further, each column is prescribed a duration in which the contained instruction set is to be performed. A bit value of 1 implies that the respective unit is switched on. Once a unit is activated, the duration of time it is instructed to its current task is determined not only by the corresponding time interval index but also by subsequent bits in its row of the instruction matrix. For example, if the following bit in a particular units' row is another 1, then the unit is prescribed only the single time interval associated with the index in which it was originally switched on. However, if the following bit is another zero, then the corresponding time interval associated with the subsequent bit is added to the total duration for which the original instruction is to be performed. This addition of time may continue until either another bit valued at 1 is reached or the current accumulated time exceeds that of the unit's maximum batch time. Should the initial time interval already exceed the unit's maximum batch time, then the unit will perform the task in the maximum time specified and move the resultant batch into intermediate storage prior to the end of this accumulated time. If in any other case, the unit could finish a task prior to the time instructed then this batch load will also move along to the relevant storage units. In the example in Table 3.3, the mixer is instructed to activate and perform a task for the duration of 4.665 hours. After 4.665 hours, the mixer is instructed to perform another batch for 3.479 hours. During this 3.479 hour interval, the reactor is activated and also produces a batch. In the third interval, the mixer is again instructed to perform a batch, however, this time

interval does not exceed the minimum time required to switch the unit on and thus the instruction is ignored. The reactor processes another batch within this interval. The third interval also sees the purifier turn on for the first time and process its first batch. In the final interval of 1.429 hours, the mixer will again ignore the instruction on account of not exceeding the minimum time of the unit. Similarly, the the interval does not allow the reactor to turn on either. Finally, the purifier processes its second batch and the example terminates with the extracted red instructions seen in Table 3.3, along with the corresponding time interval vector.

Early attempts in this research towards this framework, was to utilise -1 bit values to indicate a continuation of a task across multiple time intervals. Specifically, 1 implying the starting of a unit, -1 a continuation of the task and 0 that the unit is inactive. However, the modification to use only 0's and 1's as described above allowed a large reduction in the feasible region with respect to the instruction matrix. The drawback on this dimensionality reduction is the reliance of non-zero bit values to propose the end of a current task and the initialisation of a subsequent one. However, the implementation of the both strategies showed that the dimensionality reduction advantage reported better consistency in results at specified algorithmic parameters than of the inclusion of additional bit value of -1.

3.2.5 Motivating Example

The application of the continuous-time approach applied to the motivating example was alluded to in the aforementioned section. However, the framework is further described here.

The operations matrix, i_m, again represents the number of units and corresponding column instructions. The values of these elements are still binary since the units can each only perform a single operation. Initial instruction matrix populations are

generated randomly, with column size subject to the number of eventpoints. The time interval vector is determined by the number of eventpoints. It is assumed that $n > 3$ in this example, since at least 3 operations would need to transpire in order to produce some product. Once the number of eventpoints is specified, this time interval vector is constructed by randomly discretising the time horizon into n intervals.

As with the discrete-time approach, the representation of candidate solutions within this framework is complicit with each of the three metaheuristics allowing for each technique to explore solutions in each of their own respective means. The fitness function used obeys all constraints outlined by Kondili et al. (1993).

Application of the Genetic Algorithm

The outline of the application of the GA to the motivating example follows:

1. The population is initialised through the generation of trial candidate solutions. A population size, P, is generated, where each member of the population possess a operations matrix as well as a corresponding time interval vector.

2. The algorithm is run for a predetermined number of generations. As in the discrete-time approach, elitism is applied with the highest ranked candidates passed onto the next generation automatically. From these surviving candidates, reproduction takes place until the population is replenished. The amount of children to generate is again $P - \theta P$, where these children are generated through the crossovers applied on randomly selected elite candidates. The crossover operation is performed in the same manner as in the discrete-time approach, with the single-point crossover limited the set outlined by:

$$c_p = \{2, ..., n-1\}, \tag{3.5}$$

The crossover operation is also applied to the time intervals, albeit in a slightly different manner. The time interval elements prior to the crossover point and following the crossover point are swapped as they are with the instruction candidates. However, the elements at the exact crossover points that are handled differently. Due to the time intervals before and after the crossover point being swapped, the summation of the time intervals may now exceed or fall well short of the time horizon. Therefore, the crossover elements in each child are subjected to either a addition or subtraction of time in order to force this summation of intervals to be less than or equal to the time horizon, i.e:

$$t_{v_{c_p}} = \max\left\{ t_{v_{c_p}} + \left(H - \sum_{i=1, i\neq c_p}^{n} t_{v_i} \right), 0 \right\}, \tag{3.6}$$

constricting $t_{v_{c_p}}$ to be non-negative. Essentially, the crossover is performed and the point of crossover absorbs the time cost associated with this operation - be it an increasing or decreasing adjustment.

3. Once the population is replenished through reproduction the mutation operator is applied. The given number of mutations to perform is randomly distributed across operations matrices and time interval vectors. The total number of mutations to perform is again computed with $\psi(P-\lambda)$.

   The mutation operator applied to the operations matrices is randomly uniform between either the aforementioned *bit-flip* or regenerating a column for an arbitrary time interval.

   The mutation applied to the time interval vectors utilises neighbourhood exploration. Given some neighbourhood radius, Δ, a random element in the

time interval vector is selected for mutation. This mutation element, m_p is defined by the set:

$$m_p = \{1, 2, \ldots, n\}, \tag{3.7}$$

implying that the mutation can take place on any particular time interval. The mutation operation is defined as:

$$t_{v_{m_p}} = \max\left\{t_{v_{m_p}} \pm (r \times \Delta), 0\right\} \tag{3.8}$$

where $r \in [0,1]$ is random and uniformly generated. Therefore, the change observed by $t_{v_{m_p}}$ is bounded somewhere within the neighbourhood bound within $\pm\,\Delta$.

Similarly to the crossover applied to the time interval vectors, this mutation operator can lead to the sum of intervals exceeding or falling short of the time horizon. Therefore, the cost associated with this mutation is amortised with the remaining vector elements. Specifically, the required subtraction or addition of time is equally distributed across all other intervals, defined as:

$$\overline{t_{v_{m_p}}} = \max\left\{\overline{t_{v_{m_p}}} - \frac{\pm(r \times \Delta)}{n-1}, 0\right\}. \tag{3.9}$$

All time intervals are subjected to a lower bound of zero - meaning that in some circumstances, the summation of some vectors may still be less than the time horizon. However, this need not necessarily imply that a particular vector may produce a poor objective value. Even in such instances where a vector is severely affected by a mutation, then its it will have high likelihood of dying out on passage to subsequent generations.

4. Finally, once both the crossover and mutation operations have been

completed, the population is ready for revaluation. The above process repeats until a designated number of generations have been realised.

Application of the Simulated Annealing

The outline of the application of SA to the motivating example follows:

1. A number of initial candidate solutions are generated and evaluated, the best of which is selected for the SA process. As with the discrete-time approach, this was done to ensure that the SA was initialised with a reasonably feasible candidate solution set.

2. The algorithm runs until the initial temperature, TM, is cooled below some threshold value, ϵ. The rate of cooling applied is subject to νTM, where $\nu \in [0.9, 1)$. At any given temperature, a number (LM) of perturbations are performed. Operations that can be performed are non-uniformly sampled from either *swapping*, *inversion*, *translation* and *mutation*. These operations are described below:

    - **Swapping:** in terms of operations matrices, swaps consist of pairs of randomly selected columns exchanged positionally. This swap style can also be applied to the time vectors in which cases, randomly selected intervals are swapped positionally.
    - **Inversion:** the inversion operator is only applied to the operations matrices. Just as was illustrated in Figure 3.10 with the discrete-time approach, a portion of the operations matrix is selected, all rows and a subset of consecutive columns - which are then inverted orderly, i.e. the column indices inverted.
    - **Translation:** the translation operation is applied to both the operations matrices and time interval vectors. The operation applied to the operations matrix is exactly the same as described for the discrete-time

approach (illustrated in Figure 3.11). Similarly, the time intervals are also translated within the vector and performed index wise.

- **Mutation:** the “mutation" operator is applied to both candidate components. For the operations matrices, this mutation manifests via a bit-flip or column regeneration as described in the discrete-time approach. The mutation applied to the time intervals is the same operation that was utilised in the GA described above. The mutation applied to the time interval vector happens more frequently than any other operation, hence the aforementioned non-uniform sampling of operators.

Application of the Migrating Bird Optimisation

The outline of the application of the MBO to the motivating example is as follows:

1. The population is initialised through the generation of trial candidate solutions. The same parameters utilised in the discrete-time approach were used. The only difference between the discrete-time and continuous-time approach is that the optimal parameter, $P = 51$, applies to both operation matrices and time interval vectors, i.e. 51 trial solutions of each are generated.

2. Since the MBO utilises neighbourhood searches for solution improvement, two distinct neighbourhood search techniques are needed - one for the operations matrices and another for the time interval vectors. When invoking a neighbourhood search around any particular bird, the search will be either around the operations component of the candidate or the time intervals but never both. The reason for searching only one component at any particular instance is to localise the search as much as possible. The algorithm favours the search in the time interval component more than the operations matrix, with a probability $p = 0.66$ that the time intervals are explored versus the

operations matrix.

The neighbourhood exploration available to the operation matrices unlike the discrete-time approach, is only via the bit-flip. Since the continuous-time approach is subject to more complexity, limiting the search strategy to a more direct searching approach is preferable.

The neighbourhood search associated with the time interval vectors makes use of the same strategy employed by the GA mutation operator.

3.2.6 Primary Example

As mentioned in the discrete-time approach, within these newly introduced frameworks - the changes required from one literature example to the next are minimal. The specifications of the primary example with variable processing times are no different. The problem data is still represented in Table 3.2, the only relevant change being the processing times. Now, the reported values in Table 3.2 are mean processing times where the units' processing times vary up to 33% dependent upon the material to be processed. The representation of operations remains the same as in the discrete-time approach, with reactors 1 and 2 sampled from the integer range [0, 3] and the heater and still remaining binary.

The time interval vector representation is independent from any problem description and is generated as done in the motivating example.

Application of the Metaheuristics

When highlighting the usability of these frameworks, nothing changes from the motivating example to the primary example with respect to the application of the metaheuristics. The relevant fitness function does need to be applied and this was

the case with the fitness function upholding all constraints introduced by Kondili et al. (1993). Minor changes were required to operators invoking bit-flips on units which are not binary representations, and instead are simply resampled from their respective integer ranges. Any operators involved in the operations matrices that index according to column, again as in the motivating example, require no changes. Similarly, the time interval vector operators require no update from the motivating example.

3.2.7 Fitness Functions

Two different fitness functions were used, one for each of the literature examples. The design of both functions is the same for either the discrete-time or continuous-time approaches. Both introduced frameworks are analogous to the global eventpoint formulation seen in the MILP approaches. Therefore, the fitness functions control and measure any particular candidate solution by stepping through these eventpoints and update the status of the plant superstructure through time. Formally this is done by monitoring the volumes of all units and intermediates at each eventpoint. For example, in the case of the motivating example, the superstructure and problem data specify three units and two intermediate storage tanks with one final product. Consequently, the fitness function will update the volume of these six states (the final product included) at each eventpoint. To illustrate, the motivating example is utilised and the fitness function traversed in both the discrete-time approach and continuous-time approach. The discrete-time approach follows.

Consider a candidate solution in the discrete-time approach for the motivating example such as the candidate illustrated in Figure 3.8. At the initialisation of the fitness function call, all volume states are assumed to be empty. The superstructure itself is fed with an infinite source which is not state monitored but will fill the mixer

whenever instructed. The current eventpoint indexes into the operations matrix with its current integer value and the corresponding column is extracted. From this column the relevant instructions are attempted. If an inactive unit is indicated to be activated and should the constraints be satisfied, then the unit is activated and it will ignore any future instructions at subsequent eventpoints until the exact processing time has expired. Once this time has expired, the unit attempts to move its processed batch and is toggled inactive pending further instruction. Once the fitness function has indexed through all columns within the operations matrix, the objective value reported is the volume which reached the final volume state, i.e. the final product.

The same strategy as above is used in the primary example. However, the superstructure and problem data imply four units, four intermediate storage tanks and two final products. Therefore, the fitness function in this case tracks the updates of ten volume states as it indexes through the operations matrix.

The complexity of fitness functions required in the continuous-time framework is more elaborate. Now at the initial function call, proposed job schedules for each unit are extracted from the operations matrix and time interval vector. A job schedule for any particular unit is assigned the eventpoint integers corresponding to the start and end of instructed tasks. The fitness function then indexes through the set of eventpoints. At each eventpoint, the function checks whether any units are instructed to turn on or off and will attempt to execute these instructions if they are feasible. Again, both the motivating and primary examples make use of the same strategy.

Chapter 4

Results

The results follow in Sections 4.1 and 4.3 with the discrete-time approach and continuous-time approach respectively. All results were obtained on a AMD Ryzen™ Threadripper™ 1950x CPU @ 4GHz, with 32 GB RAM @ 2800 MHz on Windows 10 Pro 64 bit (1709 build). All experiments were run on a single CPU core, unless otherwise specified. The Genetic Algorithm (GA), Simulated Annealing (SA) and Migrating Bird Optimisation (MBO) algorithms were implemented in MATLAB R2017a and the mixed integer linear program (MILP) model on GAMS 24.8.5 r61358 with CPLEX 12.7.1.0.

4.1 Discrete-Time Approach

The discrete-time metaheuristic approach, as discussed in Chapter 3, is applied to the motivating and primary literature examples. The processing times for each unit are assumed to be fixed, i.e. the time required to process a batch is independent from the batch load. For the motivating and primary examples, the State Sequence Network (SSN) models introduced in Seid and Majozi (2012a) and Seid and Majozi (2012b) were used, as these models have been shown in the literature to be the fastest performers with regards to computational time. No further models are implemented as the aim of these results is to provide a comparison of close-to-optimal metaheuristic solutions to the fastest MILP model. The GA, SA, MBO

metaheuristics are implemented as described in Chapter 3.

4.1.1 Motivating Example

The following approach considers fixed processing times. Therefore, the model complexity is significantly less than that of its variable processing time counterparts. The reduced complexity led to a GAMS cut-off time of 10 000 seconds being set. All GAMS models are attempted to be solved to a relative gap of 0% and cases not achieving this are noted in Table 4.1. The number of eventpoints used were obtained by increasing the number of points until the first instance of a zero improvement in the objective value. Time horizons ranging from 12 hours to 168 hours are considered in increments of 12 hour intervals. The reason for this particular set of time horizons is due to the availability of comparative results for the MILP model presented in Seid and Majozi (2012a). All results pertaining to the objective values and computational results can be found in Table 4.1. Each of the three metaheuristic techniques were run for 30 trials and the CPU times for the 30 trials averaged in order to obtain a good estimation of their specific runtimes. The data pertaining to the motivating example can be found in Table 4.2. The explicit parameter values utilised for the metaheuristics in each time horizon are reported in Table 4.3.

TABLE 4.1: Table Showing Computational Results for the Motivating Example with Fixed Processing Times

Method	**Hours** (H)	**Event Points**	**CPU Time** (s)	**Relative Gap** (%)	**Success Rate** (%)	**Objective Value** (mu)
S&M	12	5	0.209	0	-	100
GA	12	12	0.056	-	100	100
SA	12	12	0.211	-	100	100
MBO	12	12	4.714	-	100	100
S&M	24	10	0.235	0	-	350
GA	24	24	0.101	-	100	350
SA	24	24	0.407	-	100	350
MBO	24	24	8.781	-	100	350
S&M	36	17	0.477	0	-	625
GA	36	36	0.203	-	100	625
SA	36	36	0.585	-	100	625
MBO	36	36	12.783	-	100	625
S&M	48	23	0.712	0	-	900
GA	48	48	0.349	-	100	900
SA	48	48	0.784	-	100	900
MBO	48	48	16.868	-	100	900
S&M	60	27	0.767	0	-	1150
GA	60	60	0.435	-	100	1150
SA	60	60	0.963	-	100	1150
MBO	60	60	20.952	-	100	1150
S&M	72	33	188.61	0	-	1425
GA	72	72	0.521	-	100	1425
SA	72	72	1.176	-	100	1425
MBO	72	72	24.993	-	100	1425
S&M	84	39	0.903	0	-	1700
GA	84	84	0.608	-	100	1700
SA	84	84	1.347	-	100	1700
MBO	84	84	28.960	-	100	1700
S&M	96	43	2.075	0	-	1950
GA	96	96	1.309	-	100	1950
SA	96	96	1.553	-	100	1950
MBO	96	96	33.101	-	100	1950
S&M	108	48	10000.556	0	-	2225
GA	108	108	1.458	-	100	2225
SA	108	108	1.742	-	100	2225
MBO	108	108	36.951	-	100	2225
S&M	120	54	1.862	0	-	2500
GA	120	120	1.609	-	100	2500
SA	120	120	1.9189	-	100	2500
MBO	120	120	40.875	-	100	2500
S&M	132	60	5.011	0	-	2750
GA	132	132	1.766	-	100	2750
SA	132	132	2.084	-	100	2750
MBO	132	132	45.058	-	100	2750
S&M	144	65	1.803	0	-	3025
GA	144	144	1.926	-	100	3025
SA	144	144	2.254	-	100	3025
MBO	144	144	49.664	-	100	3025
S&M	156	71	2.867	0	-	3300
GA	156	156	2.095	-	100	3300
SA	156	156	2.548	-	100	3300
MBO	156	156	52.854	-	100	3300
S&M	168	75	130.449	0	-	3550
GA	168	168	2.554	-	100	3550
SA	168	168	2.693	-	100	3550
MBO	168	168	57.709	-	100	3550

TABLE 4.2: Problem Data for the Motivating Example with Fixed Processing Time.

Unit	**Capacity**	**Suitability**	τ
Unit 1	100	Mixing	4.5
Unit 2	75	Reaction	3.0
Unit 3	50	Purification	1.5
State	**Storage Capacity**	**Initial Amount**	**Price**
State 1	Unlimited	Unlimited	0.0
State 2	100	0.0	0.0
State 3	100	0.0	0.0
State 4	Unlimited	0.0	1.0

TABLE 4.3: Metaheuristic Parameters for Motivating Example with Fixed Processing Time

GA*		
Hours (H)	**Population Size**	**Number of Generations**
12	100	10
24	100	15
36	100	20
48	100	25
60	100	25
72	100	25
84	100	25
96	100	25
108	150	25
130	150	25
144	150	25
156	150	25
168	200	30
SA		
Hours (H)	**Parameter**	**Value**
All	TM	1e6
All	LM	20
All	ν	0.9
All	ϵ	1e-6
MBO		
Hours (H)	**Parameter**	**Value**
All	q	51
All	κ	3
All	ξ	1
All	η	5
All	K	100

* For all time horizons, $\theta = 0.3$ and $\psi = 0.8$

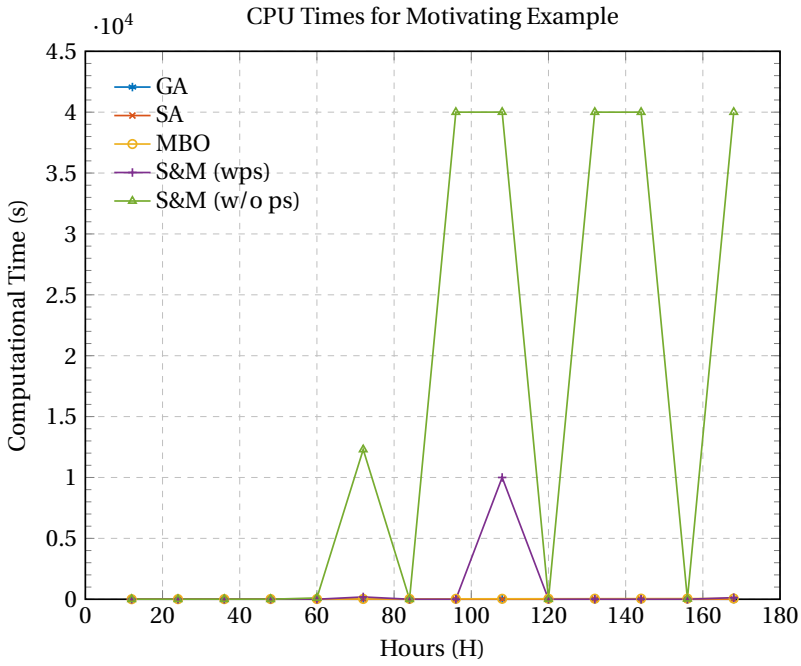

FIGURE 4.1: CPU times for Motivating Example With and Without Presolve

From the results in Table 4.1, it can see that all the metaheuristics along with the Seid and Majozi (S&M) model were able to solve the problem to global optimality with the metaheuristics having 100 % success across all trials. The overall CPU times are plotted in Figure 4.1. The CPU times for the MILP raised a number of anomalies, where the times reported did not scale as expected. The expectation being an increase in solution time with a corresponding increase in time horizon length (or problem complexity). However, at 72 hours and 108 hours, the MILP reported massive times relative to all other cases, with the model reaching the cut-off time at 108 hours. Investigating the anomaly further, the pre-solve functions in GAMS were disabled to ensure that the model was not suffering on account of pre-applied techniques. Often, the pre-solve functions in GAMS attempt to reduce the number of binary variables before initiating the deterministic search. The GAMS cut-off time was extended to 40 000 seconds in these particular cases. However, the anomaly was further observed, whereby the 120 and 156 hour time horizons solved to global optimality, while 132, 144 and 168 hour time horizons reached the GAMS

cut-off time. The anomaly is hypothesised to be due to the particular geometry of the feasible region outlined by the model in these cases. Specifically in these cases, the polytope does not lend itself to a space which can be pruned or explored rapidly as seen in the other time horizon cases. Alternatively, it may also be due to some numerical instability within the solver itself.

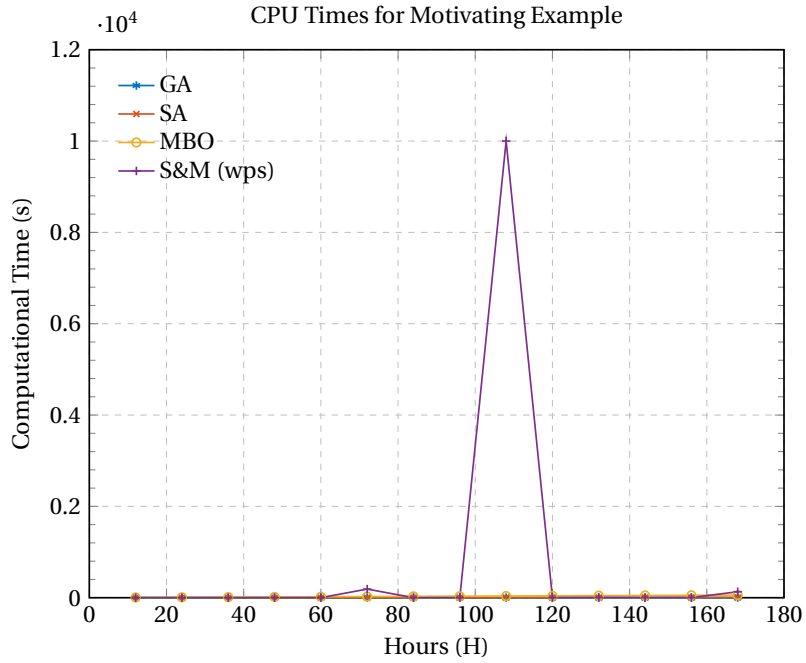

FIGURE 4.2: CPU times for Motivating Example With Presolve

The metaheuristics reported CPU times as expected, with solution times increasing proportionally with problem size. The metaheuristics are not adversely affected by the shape of the feasible region, with computational times only impacted by the size and complexity of the candidate solutions. The solution times of only the metaheuristics are plotted in Figure 4.3. It is noted that the GA was the best performer of the three, reporting fastest computational times throughout all time horizons. It is worth noting that the size of the GA had to be increased as the problem size grew in order to keep the success rate at 100%. The SA performed similar to that of the GA, albeit slightly slower. However, no increases or changes to the algorithms' parameters were needed throughout any of the time horizons. The MBO approach was substantially slower than that of the GA and SA, however, similarly to the SA,

times grew linearly with increased problem size. The slower computational times are to be expected however, as the implementation of the MBO is more complex than the GA or SA and contains operators which are not easily exploitable through vectorisable matrix operations. Interestingly, since the SA and MBO required no parameter manipulation through the different time horizons an approximate linear fit could be applied, allowing for a useful means of prediction to determine how long the method would take to solve longer time horizons.

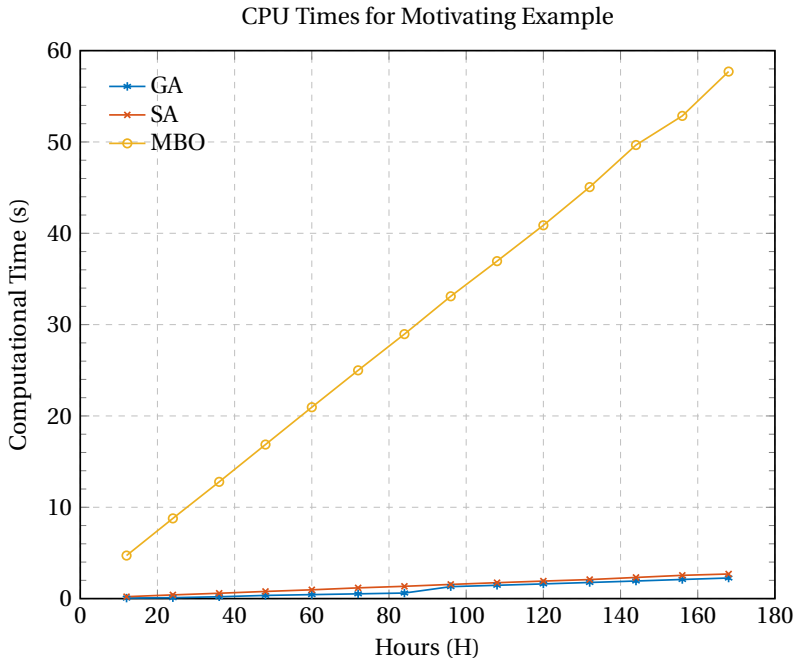

FIGURE 4.3: CPU times of Metaheuristics Only for Motivating Example

The log scale of the CPU times can be seen in Figure 4.4. The instances where the MILP exhibited the aforementioned anomalies with respect to the CPU times are noted and often increase by four orders of magnitude relative to neighbouring time horizons. The metaheuristics behaviour is smoothly described, with the MBO only growing one order magnitude over the entire set of time horizons. The SA also displayed single order growth, however, this initiated on much shorter computational times, ranging from decimals to units. The GA is the only method

which returned any “substantial" increases logarithmically. Again however, this is not inherent to the technique itself but simply due to the increased population sizes and generations required to solve the increased time horizons to 100% optimality in all trials. The Gantt chart obtained for the 36 hour time horizon is illustrated in Figure 4.5.

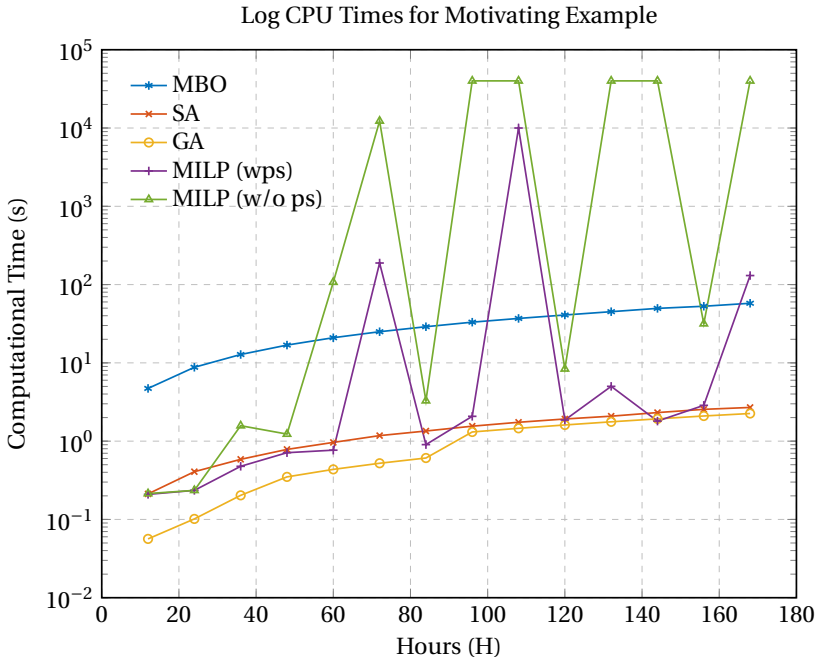

FIGURE 4.4: Log(CPU times) for Motivating Example With and Without Presolve

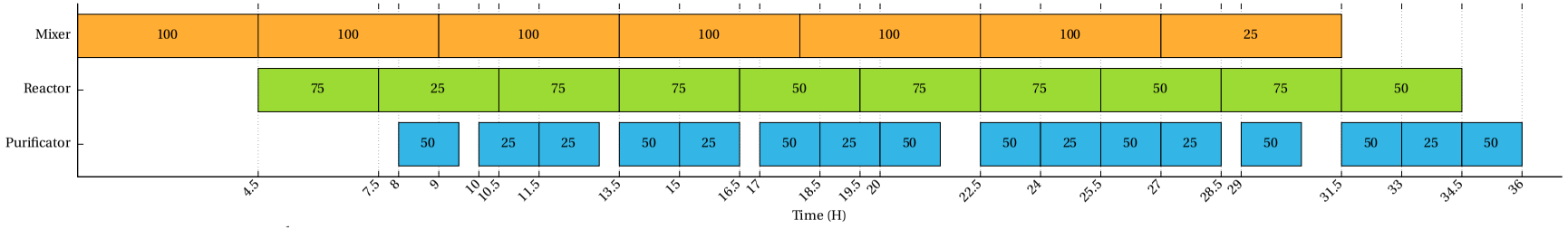

FIGURE 4.5: Gantt Chart for the Motivating Example with Time Horizon of 36 Hours

4.1.2 Primary Example

The primary literature example is also considered with fixed processing times. The GAMS cut-off time was increased to 20 000 seconds due to the primary example being more complex than the motivating example. The Seid and Majozi (2012b) MILP model was applied to each time horizon and solved to a 0% relative gap unless otherwise stated. Similarly to the motivating example, eventpoints are increased in the model until such a time that the objective value no longer improves. Time horizons were increased from 8 to 24 hours, in increments of 2 hours. All results are recorded in Table 4.4 and all parameters specific to the metaheuristics can be found in Table 4.5. Each of the three metaheuristics were again run for 30 trials with the CPU times for these 30 trials averaged to obtain an estimation of the runtime. All times are plotted in Figure 4.6. Unlike the motivating example, some metaheuristics may not find the global optimum in each trial at specific time horizons. Therefore, a second value is reported in the penultimate column of Table 4.4. The column reports the success rate of the 30 trials at finding the global optimum while the bracketed value reports a percentage difference, labelled “range". The percentage difference is the difference between the worst performing trial and the global optimum. The corresponding objective value associated with this minimum value is the bracket value reported in the final column of Table 4.4. The reported minimum value in brackets provides a lower bound for the metaheuristic at a given time horizon, and gives useful measure of dispersion of objective values attain in batch of trials.

TABLE 4.4: Table Showing Computational Results for the Primary Example with Fixed Processing Times

Method	**Hours** (H)	**Event Points**	**CPU Time** (s)	**Relative Gap** (%)	**Success Rate/Range** (%)	**Objective Value** (mu)
S&M	8	8	1.118	0	-	1917.5000
GA	8	8	0.030	-	100	1917.5000
SA	8	8	3.460	-	100	1917.5000
MBO	8	8	7.817	-	100	1917.5000
S&M	10	10	39.353	0	-	2833.7500
GA	10	10	0.070	-	100	2833.7500
SA	10	10	4.082	-	100	2833.7500
MBO	10	10	9.0418	-	26.67 (100/1.19)	2833.7500 (2800)
S&M	12	11	189.450	0	-	3638.7500
GA	12	12	0.089	-	100	3638.7500
SA	12	12	4.738	-	100	3638.7500
MBO	12	12	10.0418	-	43.33 (100/0.04)	3638.75 (3637.0833)
S&M	14	12	2257.642	0	-	4343.7500
GA	14	14	0.281	-	100	4343.7500
SA	14	14	5.339	-	100	4343.7500
MBO	14	14	11.559	-	3.33 (100/0.05)	4343.7500 (4341.6666)
S&M	16	13	11486.115	0	-	5162.0833
GA	16	16	0.308	-	100	5162.0833
SA	16	16	5.826	-	56.67 (100/0.03)	5162.0833 (5160.4167)
MBO	16	16	12.867	-	66.67 (100/0.03)	5162.0833 (5160.4167)
S&M	18	15	20104.670	4.93	-	5901.2500
GA	18	18	0.902	-	100	5901.2500
SA	18	18	9.605	-	63.33 (100/0.08)	5901.25 (5895.2500)
MBO	18	18	14.100	-	46.67 (100/0.01)	5901.25 (5900.4160)
S&M	20	16	20080.835	7.45	-	6683.7500
GA	20	20	3.568	-	100	6683.7500
SA	20	20	13.245	-	70 (100/0.99)	6683.7500 (6617.0833)
MBO	20	20	15.475	-	73.33 (100/0.99)	6683.7500 (6617.0833)
S&M	22	18	20073.621	9.33	-	7423.7500
GA	22	22	3.891	-	100	7423.7500
SA	22	22	15.545	-	63.33 (100/0.2)	7423.7500 (7408.7500)
MBO	22	22	16.660	-	16.67 (100/0.01)	7423.7500 (7422.9160)
S&M	24	21	20103.611	9.82	-	8173.3333
GA	24	24	8.895	-	100	8173.3333
SA	24	24	16.804	-	43.33 (100/0.81)	8173.3333 (8106.6667)
MBO	24	24	17.941	-	23.33 (100/0.13)	8173.3333 (8162.9160)

TABLE 4.5: Metaheuristic Parameters for Primary Example Fixed Processing Time

GA*		
Hours (H)	**Population Size**	**Number of Generations**
8	100	10
10	200	15
12	200	10
14	500	15
16	500	15
18	1000	15
20	2500	30
22	2500	30
24	4000	40
SA		
Hours (H)	**Parameter**	**Value**
All	TM	1e10
$[(8-16),\ (18-20),\ (22-24)]$	LM	[20, 45, 60]
All	ν	0.99
All	ϵ	1e-15
MBO		
Hours (H)	**Parameter**	**Value**
All	q	51
All	κ	3
All	ξ	1
All	η	5
All	K	100

* For all time horizons, $\theta = 0.3$ and $\psi = 0.8$

From Table 4.4, we can see that the S&M model was able to solve time horizons 8 to 16 hours to global optimality within the cut-off time limit, with the remaining cases reporting relative gaps from the best case of 4.93% to the worst of 9.82% respectively. While the metaheuristics offer no guarantee in obtaining global optimality, the metaheuristics did not report a superior objective value to the S&M model in any of the time horizons. The implication of no superior objective values returned by the metaheuristics, implies that the S&M model would likely converge to a 0% relative gap given sufficient time. Interestingly, the anomalies seen in the motivating example were not observed and the solution times for the S&M model increased with problem size as expected. The logarithmic growth in the solution times of the S&M model can be seen in Figure 4.8. The model grows by four orders of magnitude, at

which point the model has reached the GAMS cut-off time. It is highly likely that this exponentiation would continue further if the cut-off value was increased.

By comparison each of the three metaheuristics performed relatively well, with all three finding the global optimum within 30 trials. The GA reported 100% success rate in all time horizons. In the cases where 100% success rate was not achieved, the SA outperformed the MBO in all instances but the 16 hour time horizon. The worst lower bound for the SA was 0.99% which occurred in the 20 hour time horizon. The best lower bound for the SA was 0.03%, occurring in the case of the 16 hour time horizon. The MBO reported the worst efficiency with respect to the success rate in all time horizons, except for the 16 and 20 hour time horizons where it edged out the SA. However, the MBO did report better lower bounds on objective values than the SA on average. The best case reported a lower bound range of 0.01% in the 22 time horizon and in the worst case, a lower bound range of 1.19% in the 10 hour time horizon.

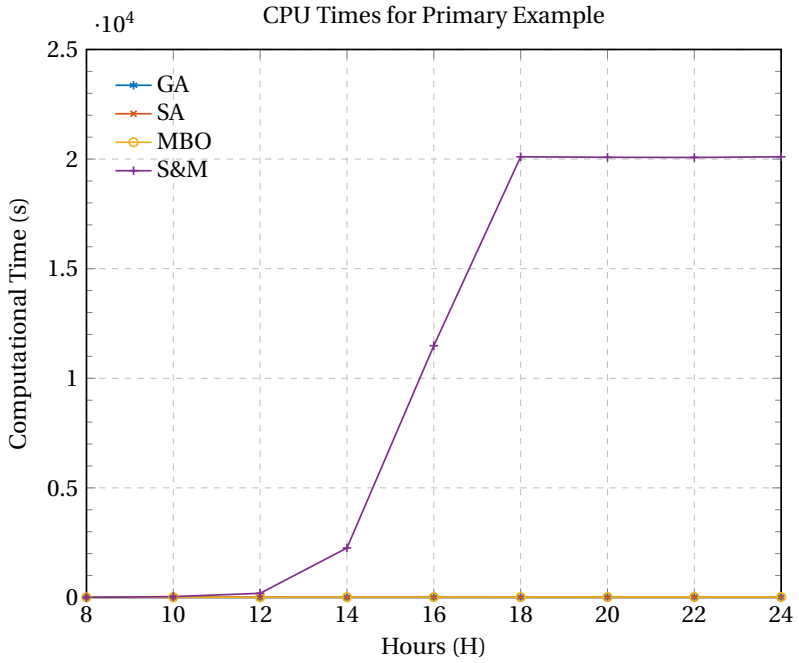

FIGURE 4.6: CPU times for the Primary Example

The metaheuristics' computational performance is illustrated in Figure 4.7. Again,

since the parameters for the MBO remain constant throughout all time horizons, a "linear" increase in the computational time is observed. The SA had a lower linear rate of increase in computational times until the 16 hour time horizon at which point the Markov chain length was increased from 30 to 45 in order to ensure that the global optimum was indeed found. A similar change in slope is observed from 22 hours onwards when the chain length was further increased to 60. As in the motivating example, the GA had the fastest performance with respect to computational times. The GA, along with the remaining metaheuristics vastly outperform the S&M model in terms of computational time. However, increased rates of computational times are observed at 14, 18, 20, 22 and 24 respectively, due to the necessary increase to the population size and the number of generations required all of which can be found in Table 4.5. In Figure 4.7 the log of CPU times for methods are plotted over the varying

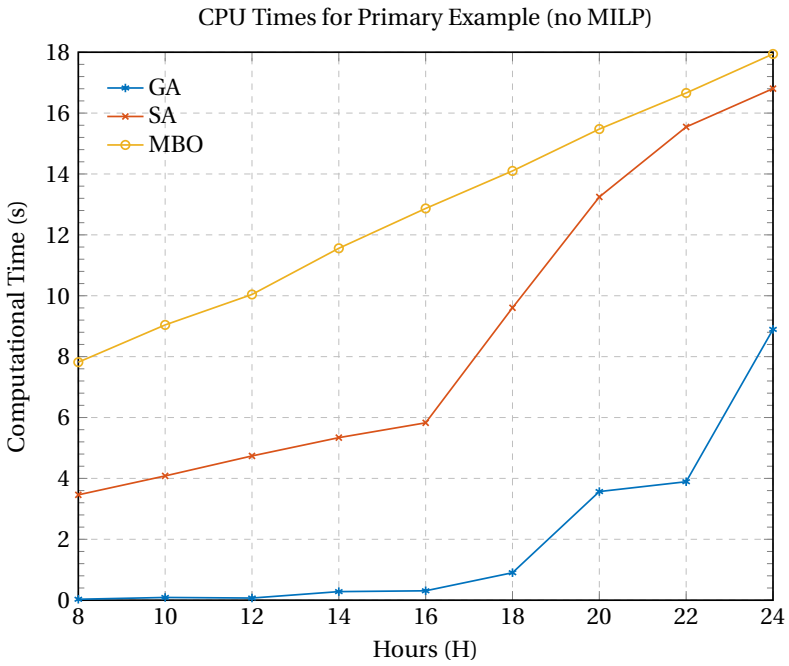

FIGURE 4.7: CPU times of Metaheuristics for the Primary Example

time horizons. The SA and MBO algorithms both increased within a single order of magnitude over the set of time horizons. The GA increased by approximately three orders of magnitude approaching a similar order to that of the SA and MBO. This increase resulting from the aforementioned population size increases, which were

done as the convergence of the GA was notably responsive to the population size. To illustrate this convergence, all cases were run to 30 generations at the relevant population sizes given in Table 4.3 and can be seen in Figure 4.9. The S&M model increases by an order of magnitude for each consecutive increase in the time horizon until the 18 hour horizon, at which point GAMS reached the cut-off time for all remaining cases.

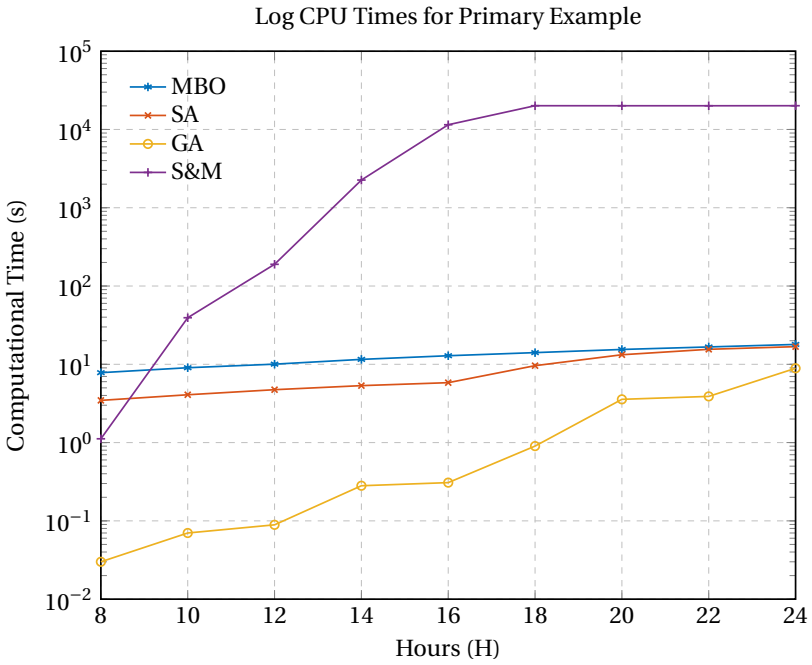

FIGURE 4.8: Log(CPU) times for the Primary Example

The best GA solution for the 24 hour time horizon is illustrated in the Gantt chart found in Figure 4.10.

4.2 Discussion

The metaheuristic approach to the motivating example with fixed processing times reports excellent computational times, even when compared to the S&M model. In fact, other than the 72 and 108 time horizons in which the anomaly was observed, the GA outperforms the S&M model in all cases except the 120 hour time horizon with respect to computational time, where it was 0.253 seconds slower. The use of an

eventpoint for every half an hour of the time horizon for the metaheuristics showed to have little impact on the computational times - a result which strongly supports its use in practical settings. Another major advantage of this metaheuristic discrete-time approach is that the correct number of eventpoints is no longer a parameter that needs to be sought and optimised. In fact, even further discretisation of the time horizon would be readily solvable using these techniques (at least using the GA and SA) allowing for the approach to be applied to plants were the highest common factor (HCF) of units is considerable smaller.

Due to the feedback loops in the superstructure of the primary example, the feasible region for the metaheuristics is considerably larger, effectively 64 possible scenarios for each additional eventpoint are added as opposed to 8 of the motivating example. However, unlike the S&M model, many of these potential possibilities are completely ignored by the metaheuristics if earlier instruction make them infeasible, reducing the amount of computation required substantially. This reduction allows for significantly shorter computational times using the metaheuristics when compared to the S&M model. For example, the 18 hour time horizon is the first time that the S&M model reaches the GAMS cut-off time. At this point, the GA obtains the global optimum in a reported time of 0.902 seconds, resulting in a computational time reduction of 99.99%. Furthermore, that the S&M model would likely have required more time in the 20 - 24 hour time horizons than the cut-off time. Even in the case of the 24 hour time horizon, the best available comparison for the S&M model, the GA still reported a computational time reduction of 99.96%.

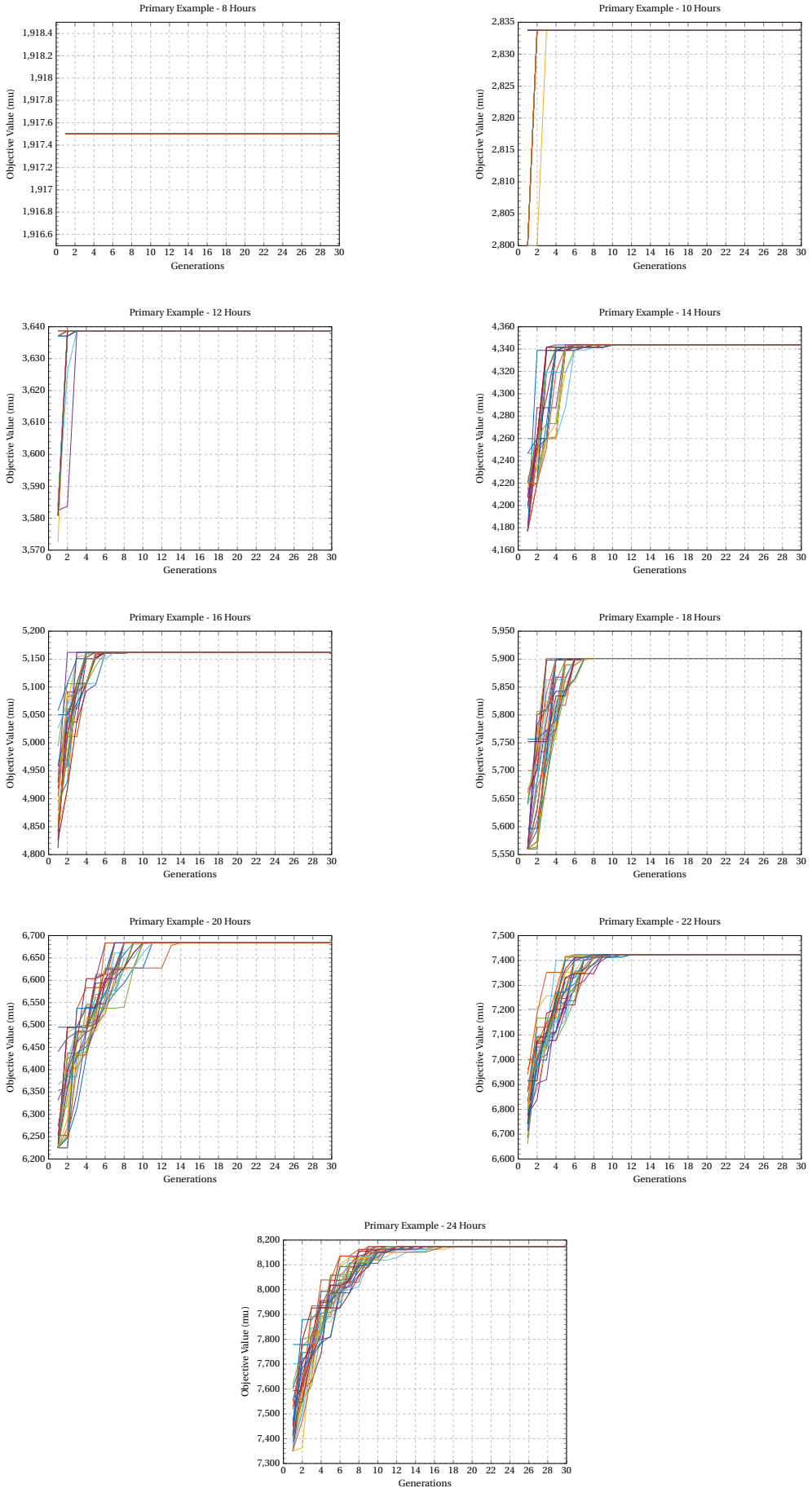

FIGURE 4.9: Convergence of GA in 30 Generations for the Primary Example

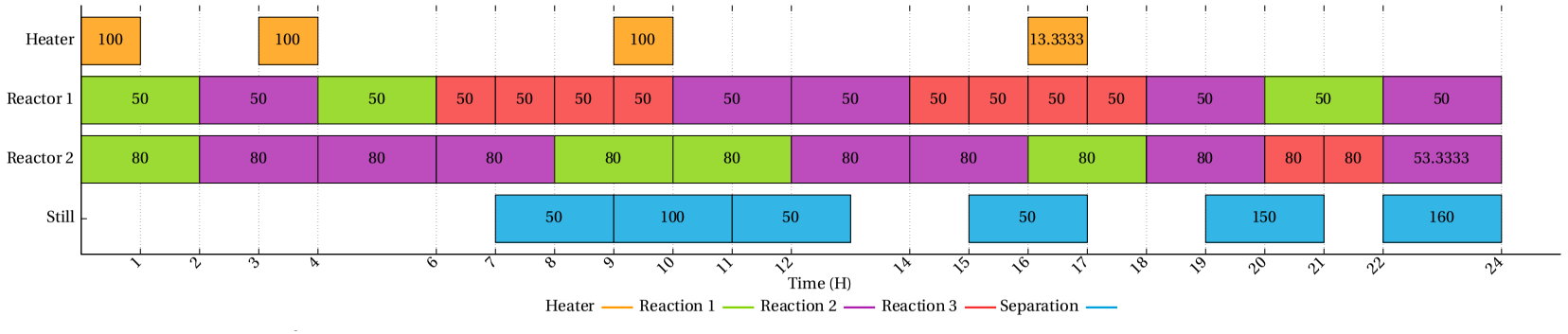

FIGURE 4.10: Gantt Chart for the Primary Example with Time Horizon of 24 Hours

4.3 Continuous-Time Approach

The continuous-time metaheuristic approach, as discussed in Chapter 3 is applied to both the motivating and primary literature examples. Now however, processing times are allowed to vary according to batch load. The same SSN models introduced by Seid and Majozi (2012a) and Seid and Majozi (2012b) are used for the motivating and primary examples respectively. The metaheuristics are implemented as described in Chapter 3. In this instance, the metaheuristic continuous-time approach is analogous to the global eventpoint based MILP models while the S&M models are unit-based eventpoint models.

4.3.1 Motivating Example

The motivating example is now considered with variable processing time. The variability is associated to the batch load to be processed, i.e. the more material loaded into units, the longer the process takes. The problem data for this example can be found in Table 4.6. All the solutions implemented obey the constraints as introduced by Ierapetritou and Floudas (1998a).

Since the problem under consideration is now that of variable processing time, the complexity of the problem has increased substantially. As result, all GAMS models were prescribed a maximum of 40 000 seconds before cut-off time in their attempt to solve the specific model to a 0% relative gap. Similar to the fixed processing times, the optimal number of eventpoints for the S&M model was determined by increasing the number of eventpoints until the first occurrence of a zero consecutive improvement in the objective value. The first eventpoint confirming this lack of improvement is the value reported in Table 4.6. Increasing time horizons are investigated, ranging from 12 hours to 74 hours, in increments of 12 hours. Similar to the fixed processing time approach, a final long-term time horizon of 168 hours was investigated. All

computational times and objective value results for these cases are reported in Table 4.7, with the CPU times plotted in Figure 4.11.

As before, each of the three metaheuristic approaches were run for 30 trials and the times for these 30 trials averaged to obtain a final approximate CPU time. The specific parameters for each method can be found in Table 4.8. Optimal parameters were not exhaustively sought, however, from the described experimentation the reported parameters proved sufficient to illustrate the suitability of the implemented metaheuristics.

TABLE 4.6: Problem Data for the Motivating Example with Variable Processing Time.

Unit	**Capacity**	**Suitability**	(α)	(β)
Unit 1	100	Mixing	3.0	0.0300
Unit 2	75	Reaction	2.0	0.0267
Unit 3	50	Purification	1.0	0.0200
State	**Storage Capacity**	**Initial Amount**	**Price**	
State 1	Unlimited	Unlimited	0.0	
State 2	100	0.0	0.0	
State 3	100	0.0	0.0	
State 4	Unlimited	0.0	1.0	

TABLE 4.7: Table Showing Computational Results for the Motivating Example with Variable Processing Times

Method	**Hours** (H)	**Event Points**	**CPU Time** (s)	**Relative Gap** (%)	**99/95/90 Percentile** (%)	**Objective Value** (mu)
S&M	12	5	0.209	0	-	71.4733
GA	12	4	6.933	-	90/100/100	71.4702
SA	12	4	5.212	-	93.33/96.67/100	71.4694
MBO	12	4	12.513	-	90/96.67/100	71.4227
S&M	24	9	0.450	0	-	249.9390
GA	24	10	8.478	-	3.33/96.67/100	247.6033
SA	24	10	6.618	-	10/100/100	245.4191
MBO	24	10	15.691	-	0/70/100	246.4199
S&M	36	15	15.074	0	-	446.9473
GA	36	14	10.150	-	60/100/100	445.4833
SA	36	14	8.063	-	3.33/90/100	442.4258
MBO	36	14	92.022	-	3.33/50/86.67	442.1303
S&M	48	19	241.672	0	-	646.9474
GA	48	23	30.570	-	10/100/100	645.2676
SA	48	23	20.336	-	0/93.33/100	636.2748
MBO	48	23	115.798	-	3.33/36.67/96.67	640.5882
S&M	60	25	12724.861	0	-	846.9473
GA	60	27	58.831	-	10/90/100	843.9086
SA	60	27	55.871	-	0/83.33/100	836.0794
MBO	60	27	127.693	-	0/23.33/96.67	825.6992
S&M	72	30	40003.179	0.95	-	1046.9743
GA	72	33	65.481	-	6.67/96.67/100	1043.1549
SA	72	33	79.558	-	0/93.33/100	1025.3750
MBO	72	33	149.722	-	0/43.33/100	1032.8482
S&M	168	57	40004.509	0.28	-	2652.7777
GA	168	76	1021.346	-	6.67/100/100	2632.3795
SA	168	76	133.431	-	0/10/96.67	2561.8789
MBO	168	76	2437.281	-	0/6.67/100	2547.8295

TABLE 4.8: Metaheuristic Parameters for Motivating Example Variable Processing Time

GA							
Hours (H)		**Population Size**		**Generations**	**Crossover θ**	**Mutation ψ**	Δ
12		500		25	0.1	0.6	0.125
24		500		25	0.1	0.8	0.125
36		500		25	0.3	0.8	0.500
48		1000		50	0.1	0.8	0.500
60		1000		50	0.1	0.8	0.250
72		1000		50	0.1	0.6	0.500
168		2000		200	0.1	0.6	1.000
SA							
Parameter	**12 H**	**24 H**	**36 H**	**48 H**	**60 H**	**72 H**	**168 H**
LM	40	40	50	80	80	100	150
ϵ	1e-3	1e-3	1e-3	1e-3	1e-4	1e-4	1e-4
TM	1e7	1e7	1e7	1e7	1e8	1e8	1e8
ν	0.90	0.90	0.90	0.90	0.95	0.95	0.95
Δ	0.25	0.25	0.25	0.25	0.25	0.25	0.25
MBO							
Hours (H)			**Parameter**			**Value**	
All			q			51	
All			κ			3	
All			ξ			1	
$[(12-24),(36-72),168]$			η			$[1,5,10]$	
All			K			100	
All			Δ			0.25	

Considering the results in Table 4.7, the S&M model does not exhibit the same anomalies that were observed in the fixed processing times case. Instead, CPU times increased as expected, excluding the first two time horizons which remained within the same order of magnitude. The observed growth in magnitude by the S&M model for the varying time horizons is illustrated in Figure 4.12. While the S&M model does perform well in the cases of 12 and 24 hour time horizons, the exponential increase in computational times begins from the 36 hour case and continues throughout. Non-zero relative gaps are reported in the cases of 72 and 168 hour time horizons with the maximum gap observed in the 72 hour time horizon, in which the relative gap was 0.95%.

The lower bound metric and range that was introduced and reported in the discrete-time approach (see Table 4.4) for the metaheuristics is replaced with another metric in the continuous-time approach. The reason for this change is due to the higher variability in objective values obtained by the metaheuristics in the primary example.

Therefore, spread of the variability in objective values obtained across a number of trials is reported. This spread is measured at 99, 95 and 90 percentile ranks, i.e. whether the objective value attained with a particular metaheuristic is within 1%, 5% or 10% of the objective value obtained by the S&M model. The success rates at each of these percentiles are reported in the penultimate column of Table 4.7. The spread provides a sense of how well the batch of trials performed overall, as well as allowing a means of measuring how well parameter changes affect the overall performance of the batch trial with respect to objective values. Although extensive exploration for optimal parameters for the metaheuristics was not undertaken (as it falls out of the scope of this research), this metric does provide a means to hone the performance of specific parameters unique to the relevant technique used.

Considering the computational performance of the metaheuristics, other than the S&M model reporting faster CPU times in the 12 and 24 time horizons, some or all of the metaheuristics report much shorter CPU times in all remaining cases. More importantly, the rate at which the computational times grow for the metaheuristics is vastly slower than that of its MILP counterpart, as seen in Figure 4.12. The GA and SA perform similarly with respect to CPU times, except for the case of 168 hours, where the increased population size and number of generations lead the GA to roughly 8 times slower than the SA as seen in Figure 4.13. However, the increased computational time in the case of 168 hours, lead the GA to have better accuracy and quality of solutions than that of the SA. The MBO was the slowest metaheuristic by some margin across all time horizons. Again, the longer computational times for the MBO are due to the nature of the algorithm as more logic is required in its implementation as opposed to the other techniques. Unfortunately, this additional time does not provide the MBO with a higher success rate at the 99, 95 and 90 percentile ranks than the GA or SA. In terms of success rates in the 99, 95 and 90 percentile ranks, the GA is the best performer. The GA reported a minimum of 90 %

in all time horizons at the 95 percentile rank and obtained objective values to within the 99 percentile of the S&M objective values in all time horizons. This is not attained by the other two techniques.

With regards to the number of eventpoints used between the metaheuristics and the S&M model, a notable difference is observed in most cases. In the 12, 24 and 36 hour time horizons, the number of eventpoints required between the S&M model and metaheuristics is similar (i.e. ± one eventpoint). However, this is due to the short time horizon implying few opportunities to actually perform tasks which results in a lower necessity for additional eventpoints. Therefore, the unit based eventpoint S&M approach should be similar to that of the global eventpoint based metaheuristics. With the increase of the time horizon, however, the deviation between the two approach styles increases. A deviation like this should occur, as a schedule would require more eventpoints to be described in the global eventpoint approach than that of the unit-based approach. In the case of the 168 hour time horizon, 19 more eventpoints were used by the metaheuristics. Noteworthy however, is that “good" solutions were often found with the metaheuristics at trials where the same number of eventpoints as the S&M model was used, but trials at these eventpoint numbers lacked sufficient consistency. This could imply that the optimal, or close-to-optimal solutions are obtainable with the metaheuristics at these eventpoint numbers, albeit, requiring a different set of optimally tuned parameters. An investigation into these optimal parameters falls outside the scope of this research, however, it will be considered in furture works. The Gantt chart obtained by the GA for the 36 hour time horizon is illustrated in Figure 4.14.

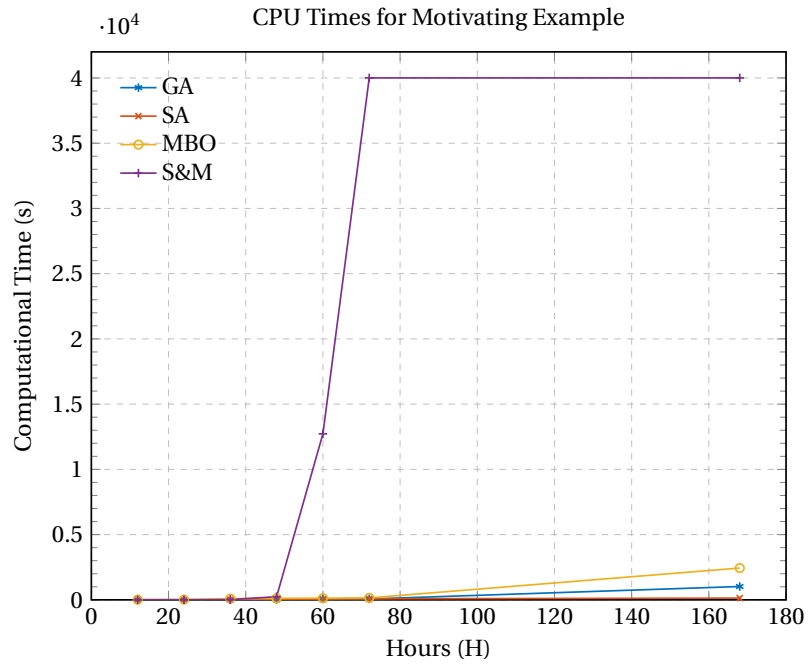

FIGURE 4.11: CPU Times for the Motivating Example

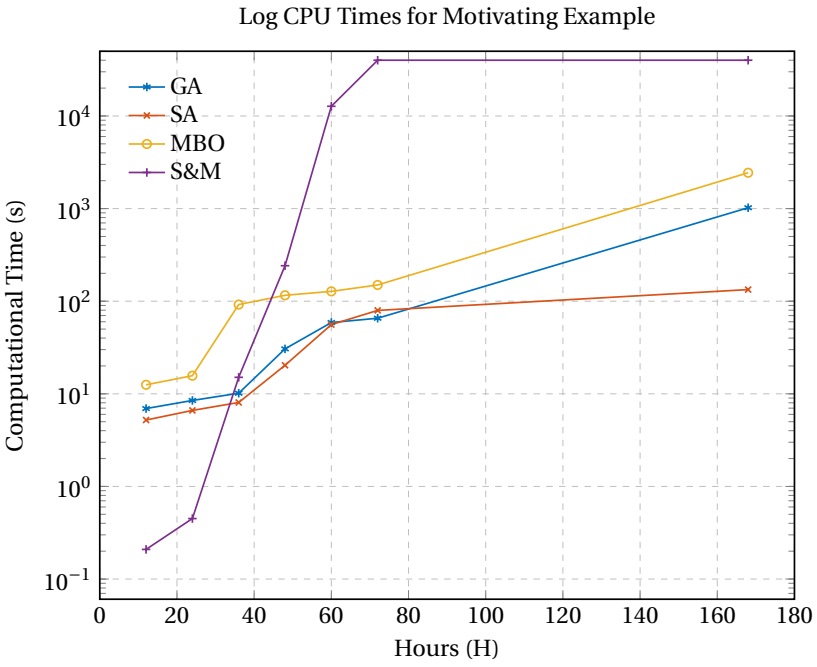

FIGURE 4.12: Log(CPU times) for Motivating Example

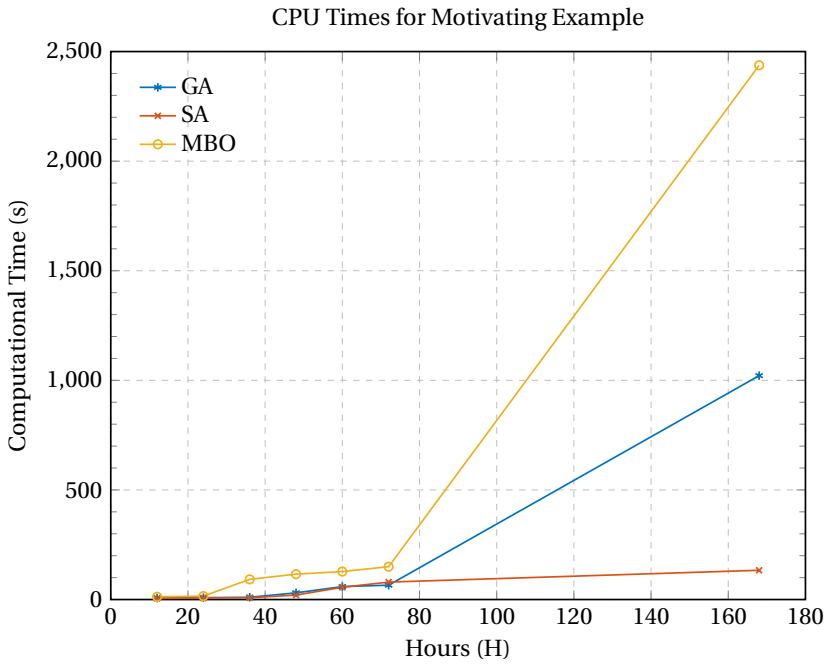

FIGURE 4.13: CPU times of Metaheuristics for the Motivating Example

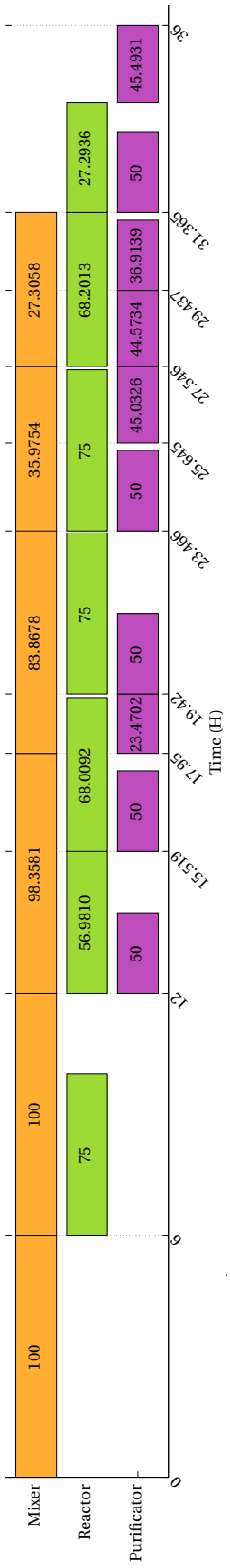

FIGURE 4.14: Gantt Chart for the Motivating Example with Time Horizon of 36 Hours

4.3.2 Primary Example

The primary example is now considered with variable processing times imposed on all units. The problem data is stated in Table 4.9. All implementations use the same constraints as outlined in Kondili et al. (1993).

TABLE 4.9: Problem Data for the Primary Example with Variable Processing Time.

Unit	Capacity	Suitability	(α)	(β)
Heater	100	Mixing	0.6667	0.00667
Reactor 1	50	Reaction 1	1.3333	0.02664
Reactor 1	50	Reaction 2	1.3333	0.02664
Reactor 1	50	Reaction 3	0.6667	0.01332
Reactor 2	80	Reaction 1	1.3333	0.01665
Reactor 2	80	Reaction 2	1.3333	0.01665
Reactor 2	80	Reaction 3	0.6667	0.00833
Still	200	Separation	1.3342	0.00666

State	Storage Capacity	Initial Amount	Price
Feed A	Unlimited	Unlimited	0.0
Feed B	Unlimited	Unlimited	0.0
Feed C	Unlimited	Unlimited	0.0
Hot A	100	0.0	0.0
Int AB	200	0.0	0.0
Int BC	150	0.0	0.0
Impure E	200	0.0	0.0
Product 1	Unlimited	0.0	10.0
Product 2	Unlimited	0.0	10.0

All results regarding the primary example are reported in Table 4.10. In obtaining the results, GAMS was again set to a cut-off time of 40 000 seconds and attempted to solve all cases to a 0 % relative gap. The optimal number of eventpoints for the S&M implementation was determined as in the motivating example, by increasing the number of points until the first instance where no further improvement was attained in two subsequent increases. Time horizons ranging from 8 to 24 hours in increments of 2 hours were investigated.

Each of the three metaheuristics were run for 30 trials and the CPU times for

the 30 trials averaged to obtain an accurate runtime. The parameters utilised for each respective technique can be found in Table 4.11. Similarly to the motivating example, optimal parameters were not obtained but sufficiently good parameters were selected through empirical investigation.

TABLE 4.10: Table Showing Computational Results for the Primary Example with Variable Processing Times

Method	**Hours** (H)	**Event Points**	**CPU Time** (s)	**Relative Gap** (%)	**99/95/90 Percentile** (%)	**Objective Value** (mu)
S&M	8	5	0.592	0	-	1498.5691
GA	8	5	11.451	-	73.33/93.33/100	1498.6346
SA	8	5	22.105	-	60/93.33/100	1498.1402
MBO	8	5	239.213	-	80/90/96.67	1498.1667
S&M	10	7	84.109	0	-	1962.6949
GA	10	8	34.386	-	6.67/83.33/100	1954.4566
SA	10	8	29.903	-	0/53.33/90	1937.9515
MBO	10	8	262.335	-	0/26.67/70	1940.1093
S&M	12	8	181.344	0	-	2658.5170
GA	12	8	36.333	-	0/80/100	2609.2722
SA	12	8	58.176	-	0/23.33/100	2610.6402
MBO	12	8	270.354	-	0/20/80	2557.0308
S&M	14	9	1580.009	0	-	3231.4351
GA	14	10	52.4712	-	43.33/46.67/100	3230.9077
SA	14	10	59.080	-	3.33/16.67/93.33	3221.8523
MBO	14	10	288.604	-	3.33/3.33/36.67	3215.5783
S&M	16	10	27015.306	0	-	3738.3800
GA	16	12	124.210	-	0/100/100	3656.7708
SA	16	12	82.102	-	0/100/100	3668.0208
MBO	16	12	311.848	-	0/20/50	3690.8115
S&M	18	13	40005.529	11.11	-	4334.8324
GA	18	14	127.338	-	0/70/100	4241.7708
SA	18	14	119.675	-	0/36.67/96.67	4261.4193
MBO	18	14	494.992	-	0/13.33/56.67	4218.7377
S&M	20	17	40007.597	12.9	-	4854.7317
GA	20	16	261.365	-	0/90/100	4786.1066
SA	20	16	181.156	-	0/63.33/100	4757.6491
MBO	20	16	540.006	-	0/16.67/73.33	4728.6477
S&M	22	19	40007.515	9.57	-	5483.5322
GA	22	18	277.997	-	6.67/73.33/100	5475.9819
SA	22	18	230.156	-	10/26.67/100	5458.9481
MBO	22	18	555.252	-	0/10/56.67	5303.6319
S&M	24	21	40005.094	11.20	-	6036.4799
GA	24	21	293.560	-	0/43.33/100	5901.2500
SA	24	21	291.348	-	0/46.67/100	5915.7516
MBO	24	21	596.370	-	0/16.67/63.33	5901.2500

TABLE 4.11: Metaheuristic Parameters for Primary Example Variable Processing Time

GA									
Hours (H)			**Population Size**			**Generations**	**Crossover** θ	**Mutation** ψ	Δ
8			500			25	0.1	0.6	0.125
10			1000			50	0.1	0.8	0.125
12			1000			50	0.1	0.8	0.500
14			1000			75	0.1	0.8	0.500
16			2000			75	0.1	0.8	0.250
18			2000			75	0.1	0.6	0.500
20			4000			75	0.1	0.6	1.000
22			4000			75	0.1	0.6	1.000
24			4000			75	0.1	0.6	1.000
SA									
Parameter	**8 H**	**10 H**	**12 H**	**14 H**	**16 H**	**18 H**	**20 H**	**22 H**	**24 H**
LM	40	50	80	80	100	150	200	200	200
ϵ	1e-3	1e-3	1e-4	1e-4	1e-4	1e-4	1e-4	1e-5	1e-5
TM	1e7	1e7	1e8	1e8	1e8	1e8	1e8	1e9	1e10
ν	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95
Δ	0.10	0.10	0.10	0.05	0.05	0.05	0.05	0.05	0.05
MBO									
Hours (H)				**Parameter**			**Value**		
All				q			51		
All				κ			3		
All				ξ			1		
$[(8-16),(18-24)]$				η			$[10,15]$		
All				K			100		
All				Δ			0.25		

The CPU times from Table 4.9 are plotted in Figure 4.15, with the logarithmic scale of these times plotted in Figure 4.16. The S&M model reports faster CPU times than all metaheuristics in the 8 hour time horizon, however, by the 10 hour case it is only faster than the MBO method, which is the slowest of all three metaheuristics techniques used. The S&M model terminates faster than the MBO in the 12 hour case as well, however, from the 14 hour time horizon onwards the computational time rapidly tends to intractability, reaching the GAMS cut-off time by the 18 hour time horizon, which occurs on all remaining cases.

With regards to the metaheuristics, all three methods outperform the S&M in terms of computational time on the medium to long time horizons (i.e. 14 hours onwards). The GA and SA again have extremely homogeneous CPU times throughout all time horizons and fall under the same orders of magnitude in all cases. The MBO approach however, is noticeable slower than the GA and SA but this difference decreases towards the longer time horizons. The reason for this is due to the parameters used. The MBO makes use of the optimal parameters suggested in

Duman et al. (2012) in all cases except the 18 through 24 hour cases. In these longer time horizons, the number of tours to perform with each leader bird was extended from 10 to 15 as experimentation showed that this had meaningful impact in the quality of the objective value obtained. This increase can been seen in the change of gradient of the MBO curve in Figure 4.16. Otherwise, Figure 4.16 shows that the MBO grows linearly with increased problem size. This linear increase in problem size is true for the GA and SA as well, however, the rate of increase appears slightly higher than that of the MBO. The oscillatory behaviour of the GA and SA curves are also due to the parameter increments required to improve the objective value solution quality at specific time horizons, although the growth rate is still linear with respect to problem scale. Overall, the GA and SA grow by one order of magnitude, the MBO remains within the same order for all time horizons. The S&M model grows six orders, which is limited again, by the GAMS cut-off and potentially would be much higher.

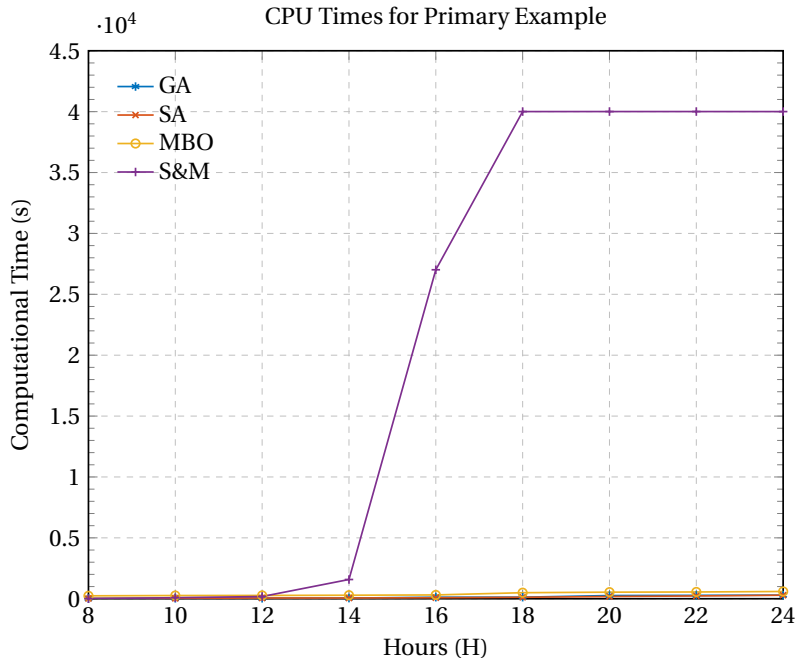

FIGURE 4.15: CPU Times for the Primary Example

In terms of the objective value solution quality of the metaheuristics, the GA again is the best overall. The GA reported 100% success at the 90 percentile rank in all time

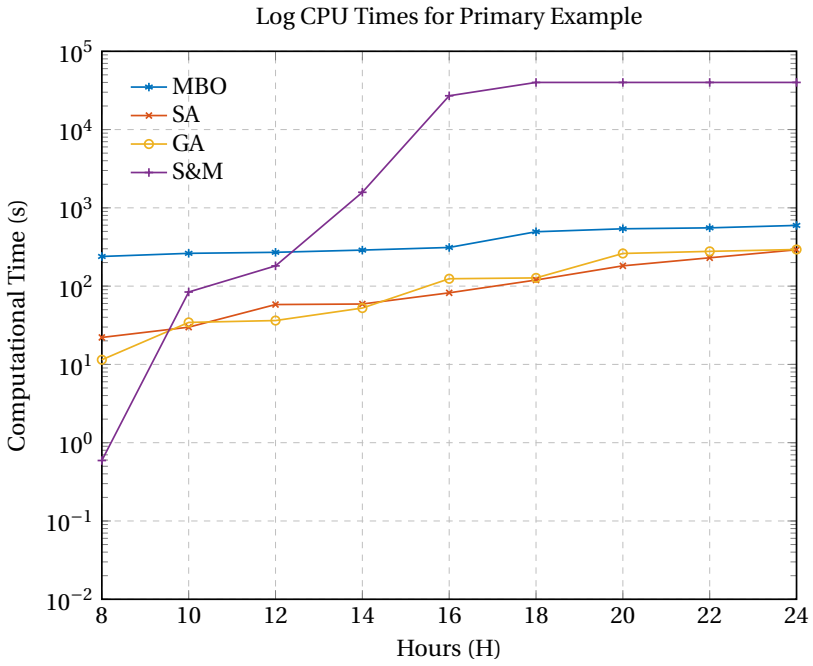

FIGURE 4.16: Log(CPU) times for the Primary Example

horizons and had strong representation in the 95 percentile rank. It also reported successful trials in the 99 percentile rank in the time horizons of 8, 10, 14 and 22 hour cases. The SA was the second best performer out the metaheuristic suite having 100% success rate at the 90 percentile in all cases except for the 10, 14 and 18 hour time horizons, in which cases the success rate was over 90%. At the 95 percentile rank, the SA had moderate success, with low success rates in the 12, 14 and 18 time horizon cases. At the 99 percentile rank, the SA managed to report a non-zero success rate in 44% of cases and outperformed the GA in objective value in the 12, 16, 18 and 24 hour time horizons. The MBO had the least success in terms of percentile ranks with less than 100% reporting in all time horizons at the 90 percentile rank. Excluding the 8 hour case, the MBO also reported less than 50% percent at the 95 percentile rank in all remaining cases and only reported a non-zero success rate at the 99 percentile rank in the 8 and 14 hour cases. A few time horizons exhibited the presence of strong local minima, however, across 30 trials all but the one strong local minimum present in the 24 hour case were overcome. The 24 hour time horizon observed a strong local value of 5901.25 which neither the GA

nor the MBO were able to improve. Interestingly, the SA did improve on this value. It is suspected that the GA and MBO converged their populations greedily to the same solutions and were unable to generate improved candidates through the operators available to the respective algorithms. In fact, it is highly likely that the operators used within the respective algorithms are the cause of this non-optimal result, where the operators force a population of candidate solutions to be overwritten with copies of trapped local optima. Specifically, it is highly likely that the GA and MBO find “good" but non optimal instruction candidates and then through mutation and neighbourhood exploration, optimise the time interval candidates around these instructions. The SA is not hamstrung in this regard as it works off a single instruction candidate and does not have a population of candidate solutions to get dominated by these substandard locals. While this is an advantage available to the SA it also implies that if an instruction candidate is extremely poor at the initialisation of the algorithm, then the algorithm may struggle to get these candidates to close-to-optimal solutions within the number of iterations dictated by the algorithmic parameters. Modifications to the GA and MBO could be implemented to overcome these issues as well. Some techniques could include random restarts as seen in Houck et al. (1996), injection strategies suggested by Nowak et al. (2015) and specific neighbourhood strategies given by De Landtsheer et al. (2018). This however, is outside of the scope of this research but should be considered in future work. In this future work, focus should be directed at optimising the methods around the time intervals. On long time horizons, the optimal solutions possess large cycles where all units are often loaded to capacity, implying time intervals are exactly the maximum time allocation required to process full batches. In these cases, metaheuristics should be adapted to move to these values immediately, rather than circle around the neighbourhood of these values. This would allow for more time along the time horizon to be allocated to other interval(s), thereby processing more product overall. This was noted commonly in both the motivating and primary examples. Both of

these literature examples display extreme sensitivity to the batch load, meaning that second or third decimals of a particular time interval, have a notable impact on the amount of material processed in said batch. In practice this is unlikely to be the case, i.e. that an apparent difference of seconds in a proposed schedule would impact the amount of material to process in a specific batch. A possible hybridisation with a slot-based strategy maybe a useful consideration in instances where the metaheuristic technique appears to approach this cyclic structure on long time horizons. An illustration of this can be seen in the Gantt chart obtained by the GA in Figure 4.17. Here, there are a number of intervals where units terminate a batch just before the next eventpoint, such as reactors 1 and 2 terminating before the time interval 10.737 hours. A slight adjustment of this eventpoint would potentially see more material processed in later batches due to this time wastage, as invariable, the longer machines are active over the time horizon, then the more product can be potential produced. The S&M and other MILP models in the literature do not exhibit this disadvantage, however, a potential hybridisation could also prove fruitful between the two approaches. For example, applying the S&M model onto the resultant GA solution to hone in on the actual global optimum. This would require a translation from the global eventpoint metaheuristic approach to a unit-based eventpoint representation to be passed as an initial starting point.

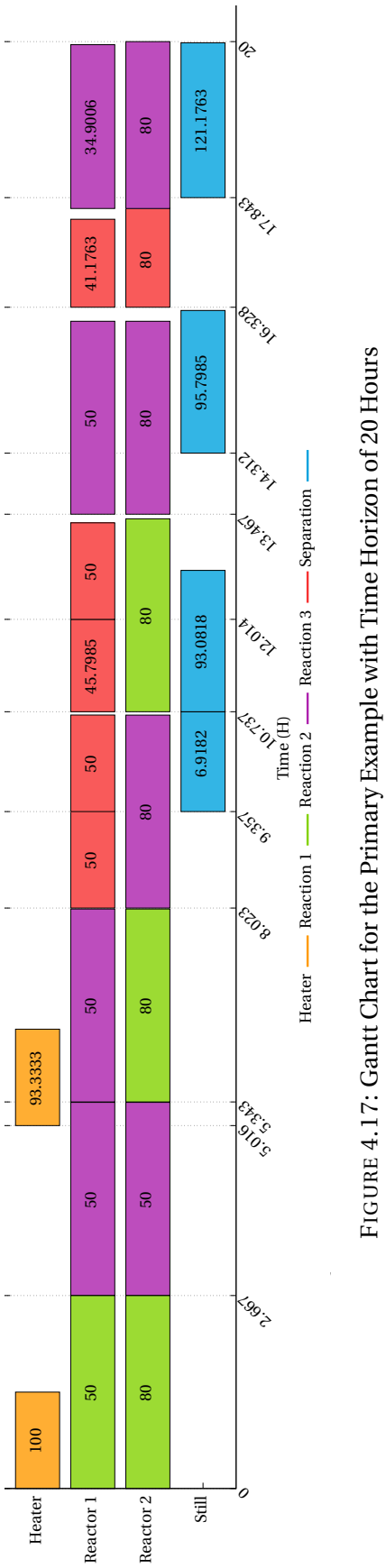

FIGURE 4.17: Gantt Chart for the Primary Example with Time Horizon of 20 Hours

4.4 Discussion

Considering the motivating example, it can be stated that the GA is able to obtain close-to-optimal solutions in computational times considerably shorter than the S&M model. Excluding the 12 and 24 hour time horizons, where the S&M model was faster, the GA reported a reduction in CPU time of 32.66% in the worst case of the 36 hour time horizon. This reduction continued in all cases with the longest time horizon of 168 hours showing a 97.44% reduction in CPU time. The SA did even better with a computational time reduction of 99.66%, albeit with a slightly worse objective value. Importantly, these percentages are based on the S&M model reaching the cut-off time of 40 000 seconds, and should the model have been run to a complete termination at a 0% relatively gap, then the computational reduction in time with the metaheuristics would be even greater. In fact, from a practical perspective using any of the metaheuristics, all time horizons could be solved within less than the amount of time required for the S&M model to solve just one of the 60 - 168 time horizon cases. A result like this significant reduction in computational time, allows plants to rapidly obtain extremely close-to-optimal schedules for numerous cases or configurations. Often, adaptability and short CPU times such as these are preferable at slightly less profitability. For example, cases where models' solution times are long enough that potential deliverables are missed or other constraints such as when tardiness may be a factor.

With the primary example, the GA was again able to provide close-to-optimal solutions in much shorter CPU times when compared against the S&M model. The GA had a total time reduction of 99.68% in the case where the S&M model first reaches the cut-off time (the 18 hour time horizon). At the longest time horizon of 24 hours, the GA still reported a total time reduction of 99.26%. The SA on average reported even further time reductions than that of the GA, however, at slightly worse objective values spreads. This should not omit the use of the SA technique as it

showed to be adept at escaping strong locals found in the feasible space and often reported improved final objective values than that of the GA. A hybridisation of the GA and SA could certainly prove useful, will the final GA solution utilised as an initial starting point for the SA. The use of the MBO is harder to justify as its success rates in the percentile ranks are poor relative to the other two metaheuristics and it is also substantially slower than the GA and SA. The speed of the MBO and the other metaheuristic techniques for that matter - is heavily dependant on the implementation. As a result, it can be expected that the metaheuristics (especially the MBO) could see considerable improvement in an optimised deployment of the algorithm such as better vectorised operations or even implementation on a low-level language such as C, or better suited high-level language such as C++. This is also an area where the metaheuristics have a clear advantage over the MILP models. Primarily, due to these models requiring implementation on frontend software packages, such as GAMS, which are linked to a suite of optimisation solvers, many of which are propriety. The only means of reducing the computational time of running one of these MILP models therefore is awaiting an update to the solver suite or using better hardware. Developments with regards to optimised implementations for these metaheuristics will be considered in future work. However, even in this prototyped implementation, the computational time reduction compared to the S&M model is markedly improved.

4.5 Parallel Implementation

One of the major advantages of the metaheuristic approach is the inherent parallelism. Since these techniques are stochastic in nature and no information need be shared between trials, theoretically there can be an infinite number of trials run simultaneously. These trials may begin at completely different starting populations and allow the algorithm to simultaneously search the space more broadly. In

addition, it was noted in some of the aforementioned results, that the GA required increases in the population size in order to improve the quality of the spread of solutions. Since the GA is completely parallelisable, this increase in population size need not take place by running a larger single instance (Fulcher, 2008) but by rather dividing this population into smaller sizes and distributing them across multiple instances of the GA running concurrently. Further, these techniques could even be instructed to share information and quality solutions across parallel searches, potentially improving the overall quality of the solutions obtained. These parallel advantages are readily available to implementations of suitable metaheuristics, but not exploitable by its MILP counterparts.

The limit of this parallelisation is solely down the specific hardware and implementation. Modern desktop CPUs are now available with up 18 cores (36 logical) allowing for massively parallel workloads. Even further, should the implementation be feasible for GPU computing, then this opens computation up to thousands of cores on standard desktop GPUs. Of course, the whole industry of cloud and high performance computing is also a possibility with effectively limitless resources available through rent based platforms such as *Amazon AWS* utilising Apache™ Hadoop® or the *Google Cloud Platform* (Zhu et al. (2016), Fazenda et al. (2012)).

To illustrate this parallelism consider the following experiment. The primary example with variable processing time is considered along the 8 hour time horizon. The time required to run this instance for 30 trials is recorded using the GA, SA and MBO methods with the same parameters used in Table 4.11. Two machines are used to run this experiment;

1. the same AMD 1950x Threadripper used to obtain the aforementioned results, with the same specification outlined in the beginning of Chapter 4.

2. A HP Z840 Workstation running an Intel® Xeon(R) CPU E5-2683 v4 @ 2.10GHZ ×64, Quadro K620 with 256 GB RAM @ 2400 MHz on Ubuntu 17.10 64 bit, GNOME 3.26.2.

Again, all thee metaheuristics were run and implemented on MATLAB R2017a. The AMD Threadripper ran the experiment utilising 1 through 10 cores and the Xeon ran the experiment utilising 1 through 30 cores. The results from the AMD Threadripper for the GA, SA and MBO are reported in Table 4.12 and plotted in Figures 4.18, 4.19 and 4.20 respectively, while the results from the HP Workstation for the GA, SA and MBO are reported in Table 4.13 and are plotted in Figures 4.21, 4.22 and 4.23 respectively.

TABLE 4.12: Table Showing Computational Results for the Primary Example with Variable Processing Times

Cores	**GA Times** (s)	**SA Times** (s)	**MBO Times** (s)
1	280.959	607.614	7539.310
2	147.338	319.860	3871.124
3	97.661	215.187	2578.231
4	78.529	173.933	2061.485
5	63.479	128.998	1597.936
6	52.590	106.486	1344.124
7	51.346	105.416	1294.247
8	43.475	88.613	1061.501
9	41.923	85.511	1067.954
10	33.242	68.298	849.985

TABLE 4.13: Table Showing Computational Results for the Primary Example with Variable Processing Times (HP Workstation)

Cores	**GA Times** (s)	**SA Times** (s)	**MBO Times** (s)
1	415.792	779.679	10272.972
2	217.837	411.369	5757.095
3	147.893	291.395	3773.373
4	119.353	224.820	2929.541
5	105.625	188.832	2284.146
6	83.435	166.605	1959.418
7	78.308	150.730	1826.187
8	66.763	122.040	1513.992
9	63.072	116.482	1484.001
11	54.577	94.404	1322.100
12	49.919	94.625	1153.567
13	49.045	92.249	1132.280
14	49.195	89.579	1097.052
15	48.759	86.780	1108.761
16	39.065	69.427	877.897
17	35.391	66.458	844.981
18	34.179	68.549	806.322
19	34.210	73.186	779.715
20	33.867	65.915	770.658
21	35.440	66.031	775.045
22	34.825	67.538	812.334
23	33.876	65.780	764.799
24	33.921	63.794	815.456
25	34.510	62.716	808.283
26	33.922	62.484	746.437
27	34.331	59.892	793.384
28	33.400	59.744	776.982
29	32.114	59.233	700.605
30	21.799	37.561	426.567

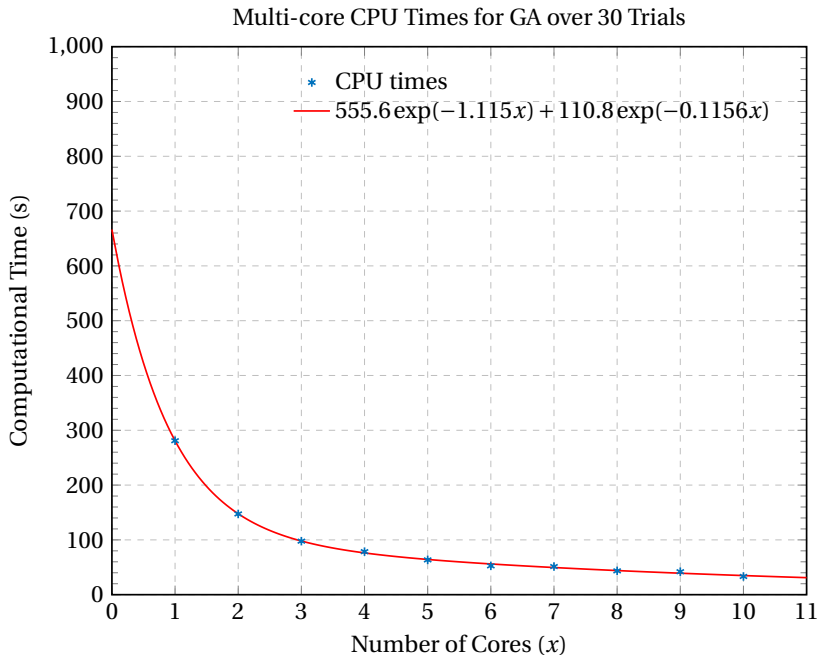

FIGURE 4.18: Time Taken to Complete 30 Trials vs Number of Cores Utilised with the GA

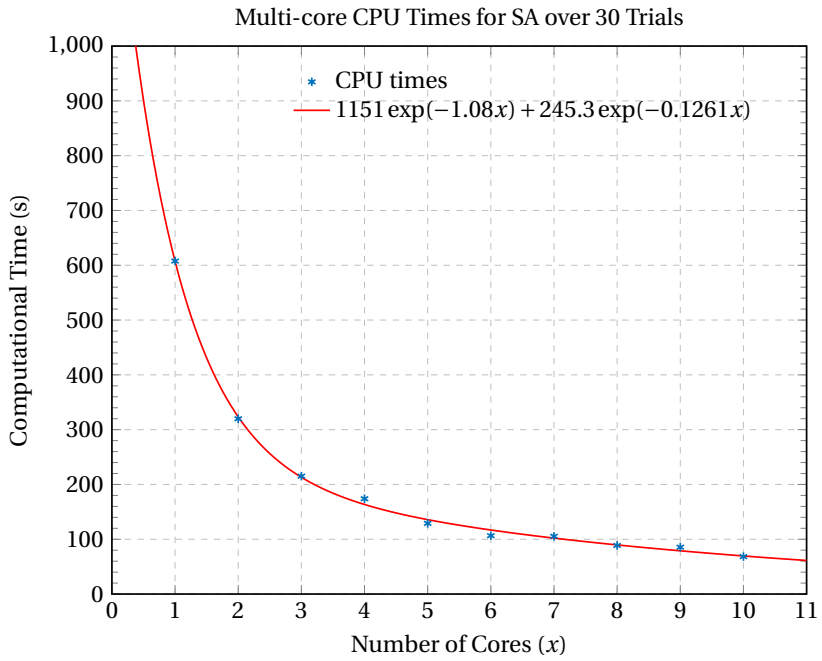

FIGURE 4.19: Time Taken to Complete 30 Trials vs Number of Cores Utilised with the SA

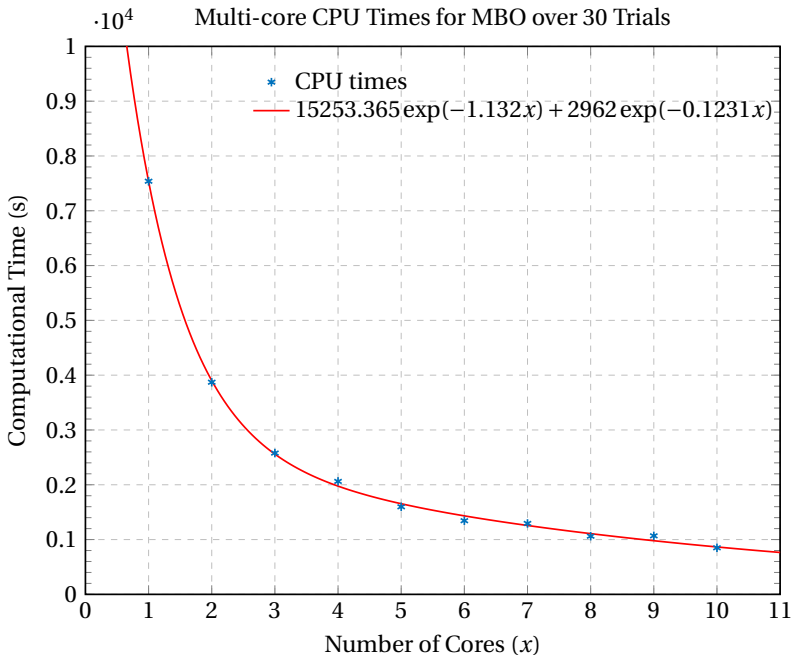

FIGURE 4.20: Time Taken to Complete 30 Trials vs Number of Cores Utilised with the MBO

From the results reported above, it can be seen that all three metaheuristics exhibit an exponential decay with respect to computational times as the number of cores increase. A feasible upper bound in terms of time reduction for the set of these experiments would be a reported CPU time for 30 trials to be equivalent to that of a single trial runtime. It is not possible on the AMD system since the maximum number of cores it was allowed to utilise was 10, and thus its specific upper bound would be 3 times the runtime. Considering the single runtime reported for the case in Table 4.10 for the GA, SA and MBO is 11.451, 22.105 and 239.213 seconds respectively, a means of measuring this computational time reduction is available.

In terms of the GA, we can see that in the case of 10 cores, the total time to complete the 30 trials was 33.242 seconds. Using 10 cores in the parallel implementation leads to perfect scaling since the upper bound would be 34.353 seconds implying that the GA performance was superlinear. Technically this upper bound should never be breached, however, due to the stochastic search strategy of the algorithm it is possible for the experiment to finish faster in some circumstances. With regard to the actual total time reduction for the GA from 1 core to 10, the time required went from 280.959 to 33.242 seconds, meaning a total time reduction of 88.16% to complete 30 trials. The SA reported a computational time of 68.298 seconds using 10 cores for the 30 trials, 2.9% slower than the upper bound of 66.315 seconds (3 times the averaged value from Table 4.9). The SA had a total time reduction from 1 core to 10 cores of 88.75%, slightly better than the GA. Finally, the MBO reported a 10 core computational time of 849.985 seconds, which is 11.84% slower than the feasible upper bound of 717.639 seconds. The MBO also was the median with respect to total time reduction of the three metaheuristics, reporting a reduction of 88.72%.
Rerunning the experiment on the HP Workstation reported the times and best fits seen in Figures 4.21, 4.22 and 4.23 respectively. The same runtime decaying behaviour is observed as was seen in the AMD system. The reason for the

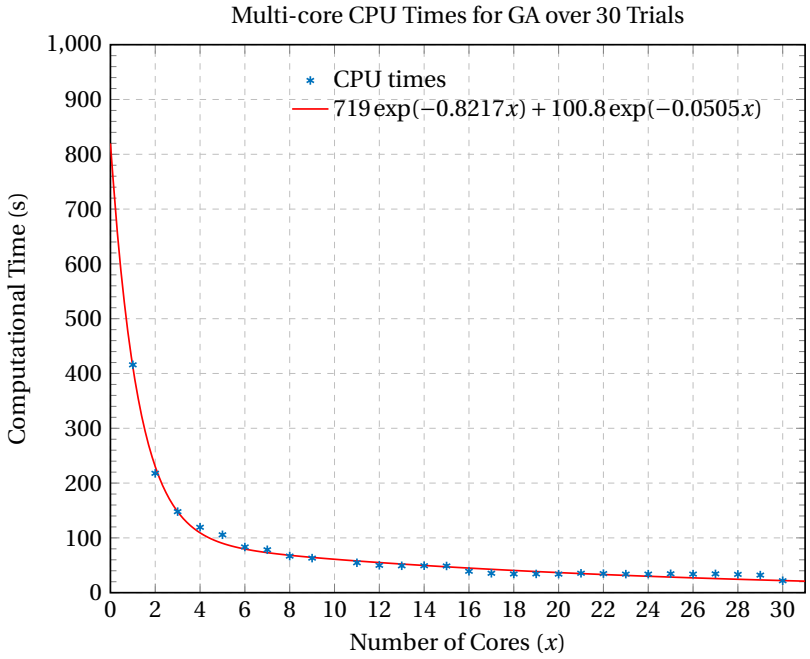

FIGURE 4.21: Time Taken to Complete 30 Trials vs Number of Cores Utilised with the GA (HP Workstation)

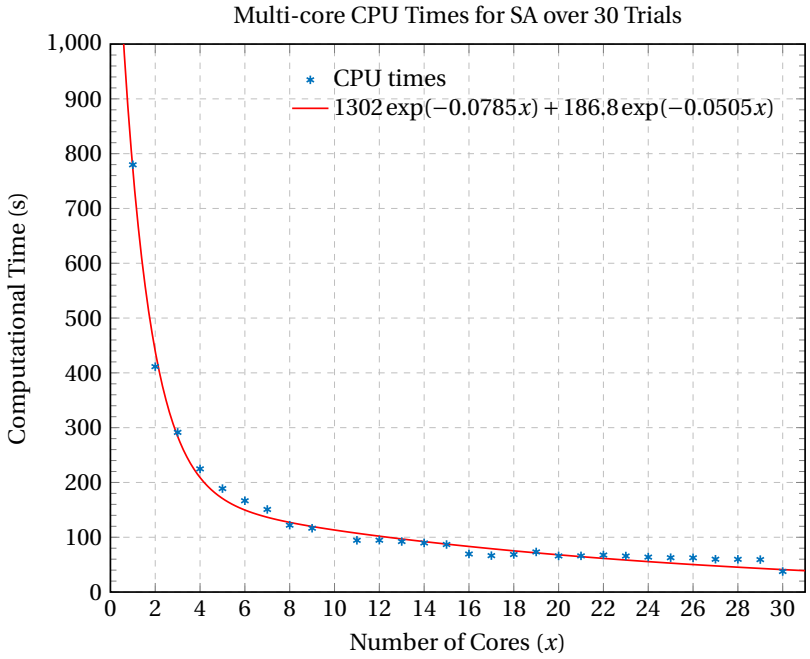

FIGURE 4.22: Time Taken to Complete 30 Trials vs Number of Cores Utilised with the SA (HP Workstation)

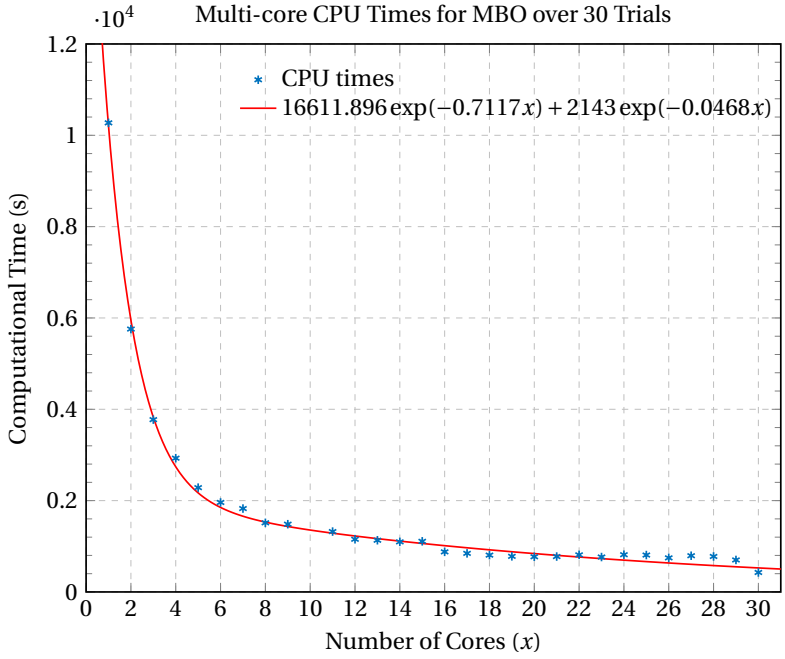

FIGURE 4.23: Time Taken to Complete 30 Trials vs Number of Cores Utilised with the MBO (HP Workstation)

implementation on the HP Workstation was to investigate the further potential reduction in total computational time for the 30 trial execution, given the addition of more cores. Increasing the core count from 10 to 30 cores yielded the following improvements;

- The GA reported a total time reduction of 94.76% up from the 88.16% achieved with AMD system.
- The SA reported a total time reduction of 95.18% up from the 88.75% achieved with AMD system.
- The MBO reported a total time reduction of 95.84% up from the 88.72% achieved with AMD system.

The results show that the MBO now reported the best overall time reduction, followed by the SA, with the GA still reporting the smallest total time reduction.

4.6 Discussion

One might be led to consider how the above metric might scale beyond the experimental results shown here. As a result best-fits are applied to the times obtained from the AMD system for the GA, SA and MBO and are plotted along with these times in Figures 4.18, 4.19 and 4.20 respectively. These best-fits yield the following equations:

$$TR_{GA}(x) = 555.6e^{-1.115x} + 110.8e^{-0.1156x}, \quad \forall\, x \in X,\ X \subseteq \mathbb{Z}, \tag{4.1}$$

$$TR_{SA}(x) = 1151e^{-1.08x} + 245.3e^{-0.1261x}, \quad \forall\, x \in X,\ X \subseteq \mathbb{Z}, \tag{4.2}$$

$$TR_{MBO}(x) = 1523.365e^{-1.132x} + 2962e^{-0.1231x}, \quad \forall\, x \in X,\ X \subseteq \mathbb{Z}, \tag{4.3}$$

where TR is the time required for the algorithm to terminate. These equations give an approximate analytical expression for the time required for the GA, SA and MBO based on the number of cores, x, provided. Equations such as these can provide engineers or plant schedulers with valuable information as they can now decided on which computations to make within given time limit constraints as the computational time required is predictable. Since a plant would generally have a single or at most a limited number of configurations in the superstructure, then it could quite easily build up these computational profiles for various combinations and utilise them on demand. For example, if the scheduling need be changed frequently due to unseen demands or deliverables, the time required to obtain a quality solution can be looked up and subsequently executed as it will be known *a priori*.

The best-fits are also applied to the results obtained by the HP Workstation, yielding

the following equations,

$$TR_{GA}(x) = 719e^{-0.8217x} + 100.8e^{-0.0505x}, \quad \forall\ x \in X,\ X \subseteq \mathbb{Z}, \tag{4.4}$$

$$TR_{SA}(x) = 1302e^{-0.0785x} + 186.8e^{-0.0505x}, \quad \forall\ x \in X,\ X \subseteq \mathbb{Z}, \tag{4.5}$$

$$TR_{MBO}(x) = 16611.896e^{-0.7117x} + 2143e^{-0.0468x}, \quad \forall\ x \in X,\ X \subseteq \mathbb{Z}. \tag{4.6}$$

The HP Workstation results show that further reduction in time is possible and further illustrates the massively parallalisable potential with the metaheuristic approaches. Further, the ability to obtain equations like those above, allow plants or engineers to determine the sufficient hardware requirements for their respective needs. Perhaps standard multi-core desktop CPU are satisfactory, or given the size and complexity of the operation, perhaps the aforementioned cloud computing options may be considered.

Chapter 5

Conclusions and Recommendations

5.1 Conclusions

This thesis presented two frameworks for scheduling of multiproduct and multipurpose batch plants. The first framework utilises a discrete-time approach for units with fixed processing times, while the second framework utilises a continuous-time approach for units with variable processing times. Both of these frameworks are implementable and solvable with a large array of metaheuristic techniques. Presently to the author's knowledge a generalised framework for metaheuristic techniques does not exist in the literature. While metaheuristics themselves have been utilised in the scheduling of multipurpose batch plants, these have either exploited the specifics of the investigated problem or lack the generality to be applied to different problems. The purpose of these developments was to address the issue of long computational times observable in the literature, primarily resultant from the currently used MILP and MINLP approaches.

While these mixed integer formulations are indeed mathematically elegant, they all suffer unavoidable intractability on time horizons longer than that of short-term production. Indeed, even on the short-term time horizons, sufficiently complex models display similar enumeration. With the advancement of semiconductor technology over the past decade, the accessibility and price of highly powerful

computational hardware has become both commonplace and affordable. With that in mind, many intractable problems arising in science and engineering have become far more computable, even through brute force approaches. The computational issues found in the scheduling of multiproduct and multipurpose are no different and are subject to this exploitation.

The proposed metaheuristic approach and corresponding frameworks were developed to accomplish this exploitation. The introduced frameworks were shown to possess the means of representing the same constraints and strategies used in multipurpose batch plants, as that of the established MILP formulations of the literature. The novel approach of these generalised frameworks allow for quick development of a variety of different batch plant configurations. These frameworks themselves are decoupled from the solution mechanism and therefore any suitable metaheuristic may be used. Decoupling the solution mechanism from the framework itself allows for the investigation into which metaheuristics are best suited to specific multipurpose batch proccesses. Currently no frameworks allowing this exist in the literature. On the application to some well-known literature examples, the introduced frameworks, along with a suite of metaheuristics were shown to produce close-to-optimal objective values when compared against a current formulation. Importantly however, the metaheuristics were able to achieve these close-to-optimal objective values in vastly shorter computational times, with a 99% reduction in certain cases.

Finally, the implemented metaheuristics were shown to possess the potential for massive parallelism within the proposed frameworks. This metaheuristic parallelism is a major advantage over its MILP counterparts, allowing for multiple areas of the feasible region to be searched simultaneously. The number of concurrent searches that can be undertaken, is limited solely by the specific hardware used. A similar

implementation such as this is not possible with the current MILP formulations.

5.2 Recommendations for Future Work

A scheduling formulation is most appropriate when the constraints and superstructure represent a real world plant as closely as possible. Since the literature examples are more explicitly for academic purposes, it would be worthwhile applying the introduced frameworks to actual industrial cases. It is suspected that the use of a discrete representation would be more akin to the actual processes taking place in industry, in which case, this metaheuristics approach could have substantial impact.

With regard to the metaheuristics themselves, it is recommended that investigation be done into the optimisation of time intervals in candidate solutions. With the cyclic structure observed in long time horizons, often time intervals should be set to maximum unit intervals, rather than waste time traversing the surrounding neighbourhoods. This issue could potentially be addressed with some form of learning strategy or computational intelligence. A further investigation into population management should also be done, as a means to avoid greedy convergence towards certain local solutions found within the feasible region. With respect to the implementation, it is advised that the algorithms and frameworks be reimplemented in a more suitable low level language, allowing for better computational time. The development of a suitable frontend GUI would also be fruitful.

Considering the cases where a plant requires variable processing times, an investigation into a metaheuristic framework which utilises a unit-based eventpoint approach should be done. This would not only allow for further improvements with

respect to computational times, but should also improve the quality of objective values obtained through the metaheuristics, as the search space could see a significant reduction.

References

[1] C. Azzaro-Pantel et al. “A two-stage methodology for short-term batch plant scheduling: discrete-event simulation and genetic algorithm”. In: *Computers & Chemical Engineering* 22.10 (1998), pp. 1461–1481.

[2] M. H. Bassett, J. F. Pekny, and G. V. Reklaitis. “Decomposition techniques for the solution of large-scale scheduling problems”. In: *AIChE Journal* 42.12 (1996), pp. 3373–3387.

[3] J.-K. Bok and S. Park. “Continuous-time modeling for short-term scheduling of multipurpose pipeless plants”. In: *Industrial & engineering chemistry research* 37.9 (1998), pp. 3652–3659.

[4] R. E. Burkard, T. Fortuna, and C. A. Hurkens. “Makespan minimization for chemical batch processes using non-uniform time grids”. In: *Computers & chemical engineering* 26.9 (2002), pp. 1321–1332.

[5] J. Cantón Padilla. *Integrated support system for planning and scheduling of batch chemical plants*. 2003.

[6] P Castro, H Matos, and A. Barbosa-Póvoa. “Dynamic modelling and scheduling of an industrial batch system”. In: *Computers & Chemical Engineering* 26.4 (2002), pp. 671–686.

[7] L. Cavin et al. “Multi-objective process design in multi-purpose batch plants using a Tabu Search optimization algorithm”. In: *Computers & chemical engineering* 28.4 (2004), pp. 459–478.

[8] J. Cerdá, G. P. Henning, and I. E. Grossmann. “A mixed-integer linear programming model for short-term scheduling of single-stage multiproduct batch plants with parallel lines”. In: *Industrial & Engineering Chemistry Research* 36.5 (1997), pp. 1695–1707.

[9] C.-l. Chen et al. “Optimal short-term scheduling of multiproduct single-stage batch plants with parallel lines”. In: *Industrial & engineering chemistry research* 41.5 (2002), pp. 1249–1260.

[10] R. De Landtsheer et al. “Combining neighborhoods into local search strategies”. In: *Recent Developments in Metaheuristics*. Springer, 2018, pp. 43–57.

[11] I. T. Dedopoulos and N. Shah. “Optimal short-term scheduling of maintenance and production for multipurpose plants”. In: *Industrial & engineering chemistry research* 34.1 (1995), pp. 192–201.

[12] E. Duman, M. Uysal, and A. F. Alkaya. “Migrating Birds Optimization: A new metaheuristic approach and its performance on quadratic assignment problem”. In: *Information Sciences* 217 (2012), pp. 65 –77. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2012.06.032. URL: http://www.sciencedirect.com/science/article/pii/S0020025512004483.

[13] A. Eiben and S. Smit. “Parameter tuning for configuring and analyzing evolutionary algorithms”. In: *Swarm and Evolutionary Computation* 1.1 (2011), pp. 19 –31. ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2011.02.001. URL: http://www.sciencedirect.com/science/article/pii/S2210650211000022.

[14] A Elkamel et al. “A decomposition heuristic for scheduling the general batch chemical plant”. In: *Engineering Optimization+ A35* 28.4 (1997), pp. 299–330.

[15] P. Fazenda, J. McDermott, and U.-M. O’Reilly. “A Library to Run Evolutionary Algorithms in the Cloud Using MapReduce”. In: *Applications of Evolutionary*

Computation. Ed. by C. Di Chio et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 416–425. ISBN: 978-3-642-29178-4.

[16] S. Ferrer-Nadal et al. "Material transfer operations in batch scheduling. A critical modeling issue". In: *Industrial & Engineering Chemistry Research* 47.20 (2008), pp. 7721–7732.

[17] C. A. Floudas and X. Lin. "Continuous-time versus discrete-time approaches for scheduling of chemical processes: a review". In: *Computers & Chemical Engineering* 28.11 (2004), pp. 2109–2129.

[18] J. Fulcher. "Computational intelligence: an introduction". In: *Computational intelligence: a compendium*. Springer, 2008, pp. 3–78.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractibility*. WH Freeman and company New York, 1979.

[20] F. Glover. "Improved linear integer programming formulations of nonlinear integer problems". In: *Management Science* 22.4 (1975), pp. 455–460.

[21] S Gupta and I. Karimi. "An improved MILP formulation for scheduling multiproduct, multistage batch plants". In: *Industrial & engineering chemistry research* 42.11 (2003), pp. 2365–2380.

[22] S. Gupta and I. Karimi. "Scheduling a two-stage multiproduct process with limited product shelf life in intermediate storage". In: *Industrial & engineering chemistry research* 42.3 (2003), pp. 490–508.

[23] Y. He and C.-W. Hui. "A binary coding genetic algorithm for multi-purpose process scheduling: A case study". In: *Chemical Engineering Science* 65.16 (2010), pp. 4816–4828.

[24] Y. He and C.-W. Hui. "Genetic algorithm based on heuristic rules for high-constrained large-size single-stage multi-product scheduling with parallel units". In: *Chemical Engineering and Processing: Process Intensification* 46.11 (2007), pp. 1175–1191.

[25] Y. He and C.-W. Hui. “Rule-evolutionary approach for single-stage multiproduct scheduling with parallel units”. In: *Industrial & engineering chemistry research* 45.13 (2006), pp. 4679–4692.

[26] J. H. Holland. “Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence”. In: *Ann Arbor, MI: University of Michigan Press* (1975).

[27] C. R. Houck, J. A. Joines, and M. G. Kay. “Comparison of Genetic Algorithms, Random Restart and Two-opt Switching for Solving Large Location-allocation Problems”. In: *Comput. Oper. Res.* 23.6 (June 1996), pp. 587–596. ISSN: 0305-0548. DOI: 10.1016/0305-0548(95)00063-1. URL: http://dx.doi.org/10.1016/0305-0548(95)00063-1.

[28] C.-W. Hui, A. Gupta, and H. A. van der Meulen. “A novel MILP formulation for short-term scheduling of multi-stage multi-product batch plants with sequence-dependent constraints”. In: *Computers & chemical engineering* 24.12 (2000), pp. 2705–2717.

[29] M. Ierapetritou and C. Floudas. “Effective continuous-time formulation for short-term scheduling. 1. Multipurpose batch processes”. In: *Industrial & engineering chemistry research* 37.11 (1998a), pp. 4341–4359.

[30] M. Ierapetritou and C. Floudas. “Effective continuous-time formulation for short-term scheduling. 2. Continuous and semicontinuous processes”. In: *Industrial & engineering chemistry research* 37.11 (1998b), pp. 4360–4374.

[31] M. Ierapetritou, T. Hene, and C. Floudas. “Effective continuous-time formulation for short-term scheduling. 3. Multiple intermediate due dates 1, 2”. In: *Industrial & Engineering Chemistry Research* 38.9 (1999), pp. 3446–3461.

[32] J. Kallrath. “Planning and scheduling in the process industry”. In: *OR spectrum* 24.3 (2002), pp. 219–250.

[33] I. A. Karimi and C. M. McDonald. “Planning and scheduling of parallel semicontinuous processes. 2. Short-term scheduling”. In: *Industrial & Engineering Chemistry Research* 36.7 (1997), pp. 2701–2714.

[34] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, et al. “Optimization by simulated annealing”. In: *science* 220.4598 (1983), pp. 671–680.

[35] E Kondili, C. Pantelides, and R. Sargent. “A general algorithm for short-term scheduling of batch operationsI. MILP formulation”. In: *Computers & Chemical Engineering* 17.2 (1993), pp. 211–227.

[36] N. Lamba and I. Karimi. “Scheduling parallel production lines with resource constraints. 1. Model formulation”. In: *Industrial & engineering chemistry research* 41.4 (2002a), pp. 779–789.

[37] N. Lamba and I. Karimi. “Scheduling parallel production lines with resource constraints. 2. Decomposition algorithm”. In: *Industrial & engineering chemistry research* 41.4 (2002b), pp. 790–800.

[38] K.-H. Lee et al. “Scheduling of single-stage and continuous processes on parallel lines with intermediate due dates”. In: *Industrial & engineering chemistry research* 41.1 (2002), pp. 58–66.

[39] M.-F. Lim and I. Karimi. “Resource-constrained scheduling of parallel production lines using asynchronous slots”. In: *Industrial & engineering chemistry research* 42.26 (2003), pp. 6832–6842.

[40] X. Lin and C. A. Floudas. “Design, synthesis and scheduling of multipurpose batch plants via an effective continuous-time formulation”. In: *Computers & Chemical Engineering* 25.4 (2001), pp. 665–674.

[41] X. Lin, S. L. Janak, and C. A. Floudas. “A new robust optimization approach for scheduling under uncertainty:: I. Bounded uncertainty”. In: *Computers & chemical engineering* 28.6-7 (2004), pp. 1069–1085.

[42] X. Lin et al. “Continuous-time optimization approach for medium-range production scheduling of a multiproduct batch plant”. In: *Industrial & engineering chemistry research* 41.16 (2002), pp. 3884–3906.

[43] P. Lissaman and C. A. Shollenberger. “Formation flight of birds”. In: *Science* 168.3934 (1970), pp. 1003–1005.

[44] T Majozi and. X. Zhu. “A novel continuous-time MILP formulation for multipurpose batch plants. 1. Short-term scheduling”. In: *Industrial & Engineering Chemistry Research* 40.25 (2001), pp. 5935–5949.

[45] T. Majozi. *Batch chemical process integration: analysis, synthesis and optimization*. Springer Science & Business Media, 2010.

[46] T. Majozi and F. Friedler. “Maximization of throughput in a multipurpose batch plant under a fixed time horizon: S-graph approach”. In: *Industrial & engineering chemistry research* 45.20 (2006), pp. 6713–6720.

[47] T. Majozi, E. R. Seid, and J.-Y. Lee. *Understanding Batch Chemical Processes: Modelling and Case Studies*. CRC Press, 2017.

[48] C. T. Maravelias and I. E. Grossmann. “New general continuous-time State-Task network formulation for short-term scheduling of multipurpose batch plants”. In: *Industrial & engineering chemistry research* 42.13 (2003), pp. 3056–3074.

[49] C. Méndez, G. Henning, and J Cerdá. “An MILP continuous-time approach to short-term scheduling of resource-constrained multistage flowshop batch facilities”. In: *Computers & Chemical Engineering* 25.4 (2001), pp. 701–711.

[50] C. Méndez, G. Henning, and J Cerdá. “Optimal scheduling of batch plants satisfying multiple product orders with different due-dates”. In: *Computers & Chemical Engineering* 24.9 (2000), pp. 2223–2245.

[51] C. A. Méndez et al. “State-of-the-art review of optimization methods for short-term scheduling of batch processes”. In: *Computers & Chemical Engineering* 30.6 (2006), pp. 913–946.

[52] T. Meng et al. “An improved migrating birds optimization for an integrated lot-streaming flow shop scheduling problem”. In: *Swarm and Evolutionary Computation* 38 (2018), pp. 64 –78. ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2017.06.003. URL: http://www.sciencedirect.com/science/article/pii/S2210650216304965.

[53] N. Metropolis et al. “Equation of state calculations by fast computing machines”. In: *The journal of chemical physics* 21.6 (1953), pp. 1087–1092.

[54] L Mockus and G. Reklaitis. “Continuous time representation approach to batch and continuous process scheduling. 1. MINLP formulation”. In: *Industrial & Engineering Chemistry Research* 38.1 (1999a), pp. 197–203.

[55] L Mockus and G. Reklaitis. “Continuous time representation approach to batch and continuous process scheduling. 2. Computational issues”. In: *Industrial & engineering chemistry research* 38.1 (1999b), pp. 204–210.

[56] L Mockus and G. Reklaitis. “Mathematical programming formulation for scheduling of batch operations based on nonuniform time discretization”. In: *Computers & chemical engineering* 21.10 (1997), pp. 1147–1156.

[57] S. Moon and A. N. Hrymak. “Mixed-integer linear programming model for short-term scheduling of a special class of multipurpose batch plants”. In: *Industrial & engineering chemistry research* 38.5 (1999), pp. 2144–2150.

[58] S. Moon, S. Park, and W. K. Lee. “New MILP models for scheduling of multiproduct batch plants under zero-wait policy”. In: *Industrial & engineering chemistry research* 35.10 (1996), pp. 3458–3469.

[59] K. Nowak, D. Izzo, and D. Hennes. “Injection, Saturation and Feedback in Meta-Heuristic Interactions”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 1167–1174.

[60] S Orcun, I. Altinel, and Ö Hortacsu. “General continuous time models for production planning and scheduling of batch processing plants: mixed integer linear program formulations and computational issues”. In: *Computers & Chemical Engineering* 25.2 (2001), pp. 371–389.

[61] C. C. Pantelides. “Unified frameworks for optimal process planning and scheduling”. In: *Proceedings on the second conference on foundations of computer aided operations*. Cache Publications New York. 1994, pp. 253–274.

[62] J. Pekny and G. Reklaitis. “PLANNING AND SCHEDULING-Towards the Convergence of Theory and Practice: A Technology Guide for Scheduling/Planning Methodology”. In: *AIChE Symposium Series*. Vol. 94. 320. New York, NY: American Institute of Chemical Engineers, 1971-c2002. 1998, pp. 91–111.

[63] J. F. Pekny and M. G. Zentner. “Learning to solve process scheduling problems: The role of rigorous knowledge acquisition frameworks”. In: *Proceedings of the Second International Conference on Foundations of Computer-Aided Process Operations, Crested Butte, Colorado*. 1993, pp. 275–309.

[64] J. Pinto and I. Grossmann. “Optimal cyclic scheduling of multistage continuous multiproduct plants”. In: *Computers & Chemical Engineering* 18.9 (1994), pp. 797–816.

[65] J. M. Pinto and I. E. Grossmann. “A continuous time mixed integer linear programming model for short term scheduling of multistage batch plants”. In: *Industrial & Engineering Chemistry Research* 34.9 (1995), pp. 3037–3051.

[66] J. M. Pinto and I. E. Grossmann. “An alternate MILP model for short-term scheduling of batch plants with preordering constraints”. In: *Industrial & Engineering Chemistry Research* 35.1 (1996), pp. 338–342.

[67] J. M. Pinto and I. E. Grossmann. “Assignment and sequencing models for the scheduling of process systems”. In: *Annals of Operations Research* 81 (1998), pp. 433–466.

[68] W. H. Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[69] G. V. Reklaitis. “Overview of scheduling and planning of batch process operations”. In: *Batch processing systems engineering*. Springer, 1996, pp. 660–705.

[70] N. Sahinidis and I. E. Grossmann. “Reformulation of multiperiod MILP models for planning and scheduling of chemical processes”. In: *Computers & Chemical Engineering* 15.4 (1991), pp. 255–272.

[71] E Sanmarti et al. “Combinatorial framework for effective scheduling of multipurpose batch plants”. In: *AIChE Journal* 48.11 (2002), pp. 2557–2570.

[72] E Sanmarti, F Friedler, and L Puigjaner. “Combinatorial technique for short term scheduling of multipurpose batch plants based on schedule-graph representation”. In: *Computers & chemical engineering* 22 (1998), S847–S850.

[73] G Schilling and C. Pantelides. “A simple continuous-time process scheduling formulation and a novel solution algorithm”. In: *Computers & Chemical Engineering* 20 (1996), S1221–S1226.

[74] G Schilling and C. Pantelides. “Optimal periodic scheduling of multipurpose plants”. In: *Computers & chemical engineering* 23.4 (1999), pp. 635–655.

[75] R. Seid and T. Majozi. “A novel technique for prediction of time points for scheduling of multipurpose batch plants”. In: *Chemical engineering science* 68.1 (2012), pp. 54–71.

[76] R. Seid and T. Majozi. “A robust mathematical formulation for multipurpose batch plants”. In: *Chemical engineering science* 68.1 (2012), pp. 36–53.

[77] N Shah, C. Pantelides, and R. Sargent. “A general algorithm for short-term scheduling of batch operationsII. Computational issues”. In: *Computers & Chemical Engineering* 17.2 (1993), pp. 229–244.

[78] N. Shah. “Planning and scheduling-single-and multisite planning and scheduling: Current status and future challenges”. In: *AIChE Symposium Series*. Vol. 94. 320. New York, NY: American Institute of Chemical Engineers, 1971-c2002. 1998, pp. 75–90.

[79] M. A. Shaik, S. L. Janak, and C. A. Floudas. “Continuous-time models for short-term scheduling of multipurpose batch plants: A comparative study”. In: *Industrial & engineering chemistry research* 45.18 (2006), pp. 6190–6209.

[80] M. Sipser. “Introduction to the Theory of Computation . Course Technology”. In: *Inc.,* (2005).

[81] E. Ulker and V. Tongur. “Migrating birds optimization (MBO) algorithm to solve knapsack problem”. In: *Procedia Computer Science* 111 (2017). The 8th International Conference on Advances in Information Technology, pp. 71 – 76. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.06.012. URL: http://www.sciencedirect.com/science/article/pii/S1877050917311857.

[82] S. Wang and M. Guignard. “Redefining event variables for efficient modeling of continuous-time batch processing”. In: *Annals of Operations Research* 116.1-4 (2002), pp. 113–126.

[83] K. Yee and N. Shah. “Improving the efficiency of discrete time scheduling formulation”. In: *Computers & chemical engineering* 22 (1998), S403–S410.

[84] M. Zentner et al. “Practical considerations in using model-based optimization for the scheduling and planning of batch/semicontinuous processes”. In: *Journal of Process Control* 4.4 (1994), pp. 259–280.

[85] X Zhang. “Algorithms for optimal scheduling using nonlinear models”. PhD. University of London, 1995.

[86] X Zhang and R. Sargent. “The optimal operation of mixed production facilitiesa general formulation and some approaches for the solution”. In: *Computers & Chemical Engineering* 20.6 (1996), pp. 897–904.

[87] X Zhang and R. Sargent. “The optimal operation of mixed production facilitiesextensions and improvements”. In: *Computers & Chemical Engineering* 22.9 (1998), pp. 1287–1295.

[88] Z. Zhu et al. “Evolutionary multi-objective workflow scheduling in cloud”. In: *IEEE Transactions on parallel and distributed Systems* 27.5 (2016), pp. 1344–1357.