A computational study of practical issues arising in short-term scheduling of a multipurpose facility

by

Zachariah Stevenson

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2019

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis focuses on two important considerations when solving short term scheduling problems for multipurpose facilities: deciding when rescheduling should be performed and choosing efficient time representations for the scheduling problems. This class of scheduling problems is of practical importance as it may be used for scheduling chemical production facilities, flexible manufacturing systems, and analytical services facilities, among others. In these cases, improving the efficiency of scheduling operations may lead to increased yield, or reduced makespan, resulting in greater profits or customer satisfaction. Therefore, efficiently solving these problems is of great practical interest. One aspect of real world implementations of these problems is the presence of uncertainty, such as in the form of new jobs arriving, or a machine breaking down. In these cases, one may want or need to reschedule operations subject to the new disturbance. An investigation into how often to perform these reschedulings is addressed in the first part of the thesis. When formulating these problems, one must also choose a time representation for executing scheduling operations over. A dynamic approach is proposed in the second part of the thesis which we show can potentially yield substantial computational savings when scheduling over large instances.

The first part of this thesis addresses the question of when to reschedule operations for a facility that receives new jobs on a daily basis. Through computational experiments that vary plant parameters, such as the load and the capacity of a facility, we investigate the effects these parameters have on plant performance under periodic rescheduling. These experiments are carried out using real data from an industrial-scale facility. The results show that choosing a suitable rescheduling policy depends on some key plant parameters. In particular, by modifying various parameters of the facility, the performance ranking of the various rescheduling policies may be reversed compared to the results obtained with nominal parameter values. This highlights the need to consider both facility characteristics and what the crucial objective of the facility is when selecting a rescheduling policy.

The second part of this thesis deals with the issue of deciding which timepoints to include in our model formulations. In general, adding more timepoints to the model will offer more flexibility to the solver and hence result in more accurate schedules. However, these extra timepoints will also increase the size of the model and accordingly the computational cost of solving the model. We propose an iterative framework to refine an initial coarse uniform discretization, by adding key timepoints that may be most beneficial, and removing timepoints which are unnecessary from the model. This framework is compared against existing static discretizations using computational experiments on an analytical services facility. The results of these experiments demonstrate that when problems are sufficiently

large, our proposed dynamic method is able to achieve a better tradeoff between objective value and CPU time than the currently used discretizations in the literature.

Acknowledgements

The financial support provided by Natural Sciences and Engineering Research Council of Canada (NSERC), Ontario Centers for Excellence (OCE) and the industrial partner in the analytical services sector are gratefully acknowledged.

I would like to thank my supervisors Ricardo Fukasawa, and Luis Ricardez-Sandoval for their guidance and support.

I would also like to thank Joseph Cheriyan and Jochen Könemann for agreeing to read this thesis.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Scheduling is concerned with how and when to execute operations to optimize a chosen objective such as maximizing profits, or minimizing costs, subject to operational constraints such as deadlines that must be met, or available resource limitations. It is common practice for many industries to make use of scheduling as an optimization problem to guide their progress and meet various economic objectives [14, 29, 39, 40, 43]. Proper scheduling can greatly increase the efficiency of a production plant and therefore is of great practical importance. Scheduling operations over a relatively short period of time, such as a day, a shift or a week is referred to as *short term scheduling*. We are interested in scheduling in the context of the short term scheduling of a multipurpose plant in the analytical services sector.

The analytical services sector is focused on carrying out analyses on samples that are ordered by clients for various purposes, e.g. performing a nutritional analysis on a food item to create the nutritional facts panel before bringing the product to market, or performing air quality analyses to check for hazardous materials such as asbestos. Companies in the analytical services sector may receive on the order of thousands of samples on a daily basis to be processed at their facility and as such require a suitable method of scheduling operations.

A *multipurpose plant* is a facility which has a set of *processes*, such as machines or workers, that may carry out a variety of *tasks*, i.e. the plant may serve more than a single purpose, such as producing several different products. The problem of scheduling a multipurpose plant can be seen as a variant of the job shop scheduling problem [4, 32]. Jobs arrive at a multipurpose facility, and each job has a set of samples that comprise it and a sequence of processes that the samples must be processed by in order. This sequence

of processes is called the *path* of a job. The goal is to generate a schedule for the facility, which dictates what samples to assign to which processes over the length of time that is to be scheduled such that an objective is optimized. This schedule must also abide by various operational constraints of the problem such as process resource limitations and material balance constraints.

Throughout this thesis, we model these scheduling problems as mixed integer linear programs (MILP), meaning that we restrict the objective function, and constraints of the model to be linear, and allow some variables to be integral. As such, there are already a number of developed techniques designed to solve these problems, such as branch-and-bound methods, and cutting-plane methods. We do not discuss integer programming or these solution methods here, but instead refer the reader to [6] for further information. There are several different MILP scheduling models that exist and one of the key classifications of these models is the time representation that is used [24]. For this reason, we discuss this topic in more detail below.

1.1 Time Representations

An important aspect to consider when modeling these scheduling problems is the time representation that is used for scheduling operations. The time representation determines when operations may be scheduled, and can play a large role in determining the computational cost of solving the model and the final solution quality [22, 24, 45]. We call each point in time where an operation may be scheduled a *timepoint*. There are two main classes of time representation: continuous time representation and discrete time representation [13, 42].

Continuous time representations allow events to happen at precise points in time during the scheduling horizon, with the selection of where these points should be placed being decided by the model [27, 36, 53]. Because the model is able to choose precisely where timepoints should be located, we may ensure that we obtain the best solutions using this representation [24]. However, one must provide the model a fixed number of timepoints to allocate as input. If the model is allowed to allocate too few timepoints, solution quality may decrease, however if the model is allowed to allocate many timepoints, CPU time may drastically increase. In general, selecting a suitable number of timepoints to allocate for the model can be difficult.

Discrete time models instead fix the timepoints at which scheduling decisions may be made to a subset of points during the scheduling horizon, *a priori* [21, 38]. We call the

set of timepoints that the schedule may use the *time grid*. The difficulty with this representation is in choosing a suitable time grid to provide the model. There is once again a tradeoff between how coarse or fine the discretization used is, the quality of the resulting schedule and the amount of CPU time required to solve the model. Despite the continuous time formulation allowing the model to use precise points in time, multiple works have shown that using a discrete time representation results in better performance than using a continuous time representation for scheduling multipurpose facilities [22, 26, 42]. Therefore, throughout this thesis we concern ourselves with models using a discrete time representation. Moreover, among discrete time representations, the non uniform discrete time representation has been shown to perform best for the scheduling problem considered in ths thesis [22], and so we use this time representation throughout our experiments. This will be discussed further in Chapter 2.

1.2 Rescheduling

Typically, works on scheduling have considered finding an optimal schedule in a static environment where all of the operating conditions are known with certainty throughout the scheduling horizon. While it is necessary to first understand scheduling in this context, in practice it may not always be realistic to assume that all of the information pertinent to generating an optimal schedule will be known in advance, and that no unexpected disruptions to the schedule will occur during operation [49]. In particular, an initially generated schedule can become infeasible or non-optimal because of uncertainties, such as a machine breaking down [2, 5, 12, 19, 25, 34, 37], the arrival of rush orders [2, 19, 25], or actual processing times differing from expected processing times [25, 31]. Furthermore, the arrival of new job orders are considered in this thesis and are an important consideration as they may change the optimal schedule and taking into account these new arrivals may improve plant performance.

One possible way to deal with uncertainties is to reschedule, that is, to re-optimize when disruptions occur or at specific (user-defined) time intervals. The decisions involved with rescheduling can be divided into two issues of "how-to" and "when-to" reschedule [35]. The "how-to" addresses how new schedules should be generated. Some examples include a full rescheduling of all operations or a partial rescheduling, where some operations that were scheduled previously remain fixed [35, 49]. We do not focus on how new schedules should be generated in this thesis and simply assume that when rescheduling is done a new schedule is generated from scratch. This was done so that we may ensure that newly generated schedules obtain the greatest performance given the available information,

assuming that the facility may pivot to the new schedules without performance loss. If subsequent schedules must be similar to current schedules, then a partial rescheduling of operations may be preferred.

Instead, we choose to focus on when to reschedule operations in the first part of this thesis. There have been a number of works conducted over the last few decades investigating how rescheduling frequency affects schedule performance. There have been studies conducted in a variety of environments such as job shop scheduling [2, 5, 28, 37, 47, 48], chemical production scheduling [15, 20], material requirements planning [52], scheduling flexible manufacturing systems [19, 31, 34, 35], and scheduling hospital operations [17, 41, 50]. However, in most of these studies the plant and operating parameters (e.g. plant load or plant capacity) under consideration are fixed, and it is not discussed how the results and conclusions vary when these parameters are modified. Understanding the relationship between choosing when to reschedule and the operating parameters of a plant may be of practical importance for several reasons such as seasonal industries where demand may fluctuate greatly depending on the time of year, or facilities where the number of available resources may change (e.g. purchasing additional machines or laying off workers). Indeed, the results show that under certain conditions, we (counterintuitively) find that less frequent rescheduling performs better than frequent rescheduling.

Aside from rescheduling, we may consider robust or stochastic programming as a method to mitigate the effects of these uncertainties on our schedules. Robust and stochastic programming aim to find schedules which perform well under the expected disruptions that may occur (stochastic) or subject to the worst case disruptions (robust). Since the uncertainty considered in this work is the arrival of new jobs for which not much information is known (e.g. number of samples, time of arrival, or path of the job) and there does not seem to be an accurate way of predicting this information, using these methods seems to be quite challenging. Due to these challenges and time limitations, these methods were not considered in this work but could be explored for future work. Regardless of this, we must first understand the relationship between when to reschedule and plant operating parameters in the simplest setting before exploring robust or stochastic methods and this thesis aims to address this gap.

1.3 Tailoring Time Grids

Given that the results concerning how often to reschedule show that in some cases less frequent rescheduling performs better than more frequent rescheduling, large models may be required for obtaining one's desired scheduling performance. Therefore, the question

naturally arises: how do we solve these *large* scheduling problems more efficiently? This question prompts us to take a look at timepoint representations in more detail. In general, if we knew exactly which timepoints were needed to achieve the optimal solution obtained by using a continuous time representation, we would be able to greatly reduce the size of the model by using only these necessary timepoints. However, it is unlikely that we know this information beforehand, yet we must still choose a time grid to use for the model.

As mentioned previously, choosing which timepoints to include in the time grids is in general not obvious and can greatly impact the performance of the model. Furthermore, there have been a very limited number of works that have considered how to tailor time grids for individual problem instances [3, 45]. Here we mean an *instance* of a problem to be a problem with given input data, as opposed to the general problem itself.

Velez and Maravelias show in [45] that they are able to generate non uniform time grids based on the instance input data such that the optimal solution obtained by using these time grids has equal objective value to the optimal solution obtained by using an arbitrarily fine uniform discretization. Furthermore, they show that this method of choosing non uniform time grids leads to a much smaller problem size than that obtained by using a "super-fine" uniform discretization. However, to ensure that this guarantee holds, their algorithm may include many timepoints making the resulting model very computationally taxing to solve, especially for large problem instances.

In [3], Boland et al. provide a method for iteratively refining the set of timepoints that they consider for solving the service network design problem. They take a different approach than that of Velez and Maravelias, by beginning with only a few timepoints initially and then determining where to add timepoints based on the obtained solution from solving a relaxation of their problem. They are able to show that their method will also terminate with an optimal solution whose objective value is equal to that obtained by using an arbitrarily fine uniform discretization. Moreover, they show that their method performs well in practice through a computational study.

One possible solution to determining what time grid to provide the model is to solve many instances of the scheduling problem of interest in an attempt to discern a suitable initial time grid based on the optimal schedules of these instances. This may be a time and labor intensive process however, and moreover may provide bad results when optimizing an instance which does not resemble the set of instances used to determine the time grid. Furthermore, this procedure would need to be done for each application individually. Instead, the approach that we take in this work is to iteratively refine an initial discretization. By carefully choosing the initial discretization and timepoints to add between iterations, we are able to achieve a better tradeoff between solution quality and CPU time on large

problems than that obtained by using other time grids.

1.4 Contributions of the Thesis

As we have identified above several decisions that impact the quality of generated schedules and the computational difficulty of solving these models, we now outline the contributions of this thesis.

Firstly, a study investigating when to reschedule subject to varying plant parameters is carried out to address the gap in the literature. In this study we consider the rescheduling of a multipurpose facility in the analytical services sector that receives new jobs on a daily basis. Based on the results of the computational experiments, we identify some key parameters that influence how often rescheduling should be done. This study shows that care must be taken to consider not only the scheduling problem being solved, but also the operating conditions of a facility when deciding how often to reschedule. In particular, we find surprising results when the facility is starved for processing resources which can provide valuable insight into choosing a rescheduling scheme in real world scenarios.

Secondly, we present a generalized framework for iteratively refining time grids for scheduling multipurpose facilities. We propose several heuristics for deciding where to add and remove timepoints from the scheduling model's time grid between iterations. The efficacy of this framework is evaluated through computational experiments comparing its performance against the currently used time grids for scheduling discrete time multipurpose plants in the literature. The results of these experiments show that our method may obtain significant performance improvements over the current time grids in use without a substantial increase in CPU time. Moreover, since the performance of the proposed framework remains relatively stable over a range of problem sizes in our experiments, this framework provides a stepping stone toward improving instance-agnostic methods for choosing time grids.

Note that we do not provide theorems and theoretical guarantees in this thesis. Rather, the focus is on developing heuristic approaches to our scheduling problem with empirical evidence obtained through computational experiments which demonstrate that our heuristics seem to perform well in our test cases.

1.5 Structure of the Thesis

The structure of this thesis is as follows. In Chapter 2 we discuss some necessary background information on the work presented in Chapters 3 and 4 in more detail. In Chapter 3 we present our study on deciding when to reschedule subject to varying plant parameters. This includes extensive long term computational experiments to evaluate our rescheduling policies. Our work on iteratively refining the set of timepoints to schedule over is presented in Chapter 4 which includes the proposed framework, our heuristics for selecting timepoints, and computational experiments demonstrating the efficacy of our method. Concluding remarks and future work considerations are presented in Chapter 5.

Chapter 2

Background and Literature Review

This chapter includes the necessary background information and gives context to the results of Chapters 3 and 4. We begin by discussing the scheduling problem that is used throughout this thesis in section 2.1. To help develop the model for our scheduling problem, we present the definition of time layered graphs in section 2.2. In section 2.3 we present the model that is used throughout our experiments. We then discuss various strategies that are used to determine when to do rescheduling in section 2.4. In section 2.5 we review the developments with respect to choosing discrete time representations.

2.1 Problem Description

We now discuss the scheduling problem used throughout this thesis in more detail. The problem under consideration is a variant of the job shop problem. A facility receives a set of jobs to process, I, and has access to a set of processes, V. Each job $i \in I$ is made up of a discrete collection of samples, and a sequence of processes, $\Pi(i)$, called a path, that must be performed (in the specified order) on the samples. Using this notation, we specify the k'th process in the path of job i as $\Pi(i,k)$ for $k = 1, \ldots, |\Pi(i)|$. Each process $u \in V$ of the plant has a set of identical resources (e.g. machines or workers) that may perform a single operation on samples. Each of these resources u has a fixed capacity, $\kappa(u)$, measured in terms of samples and a fixed processing time, $\tau(u)$. Furthermore, we let $\rho(u)$ represent the number of resources available for each process u. We assume that resources may not be interrupted once they have started processing (no pre-emption), and that each resource may process samples from many jobs simultaneously (batch mixing), as long as there is available capacity. Additionally, samples from a single job do not need to be processed together, they

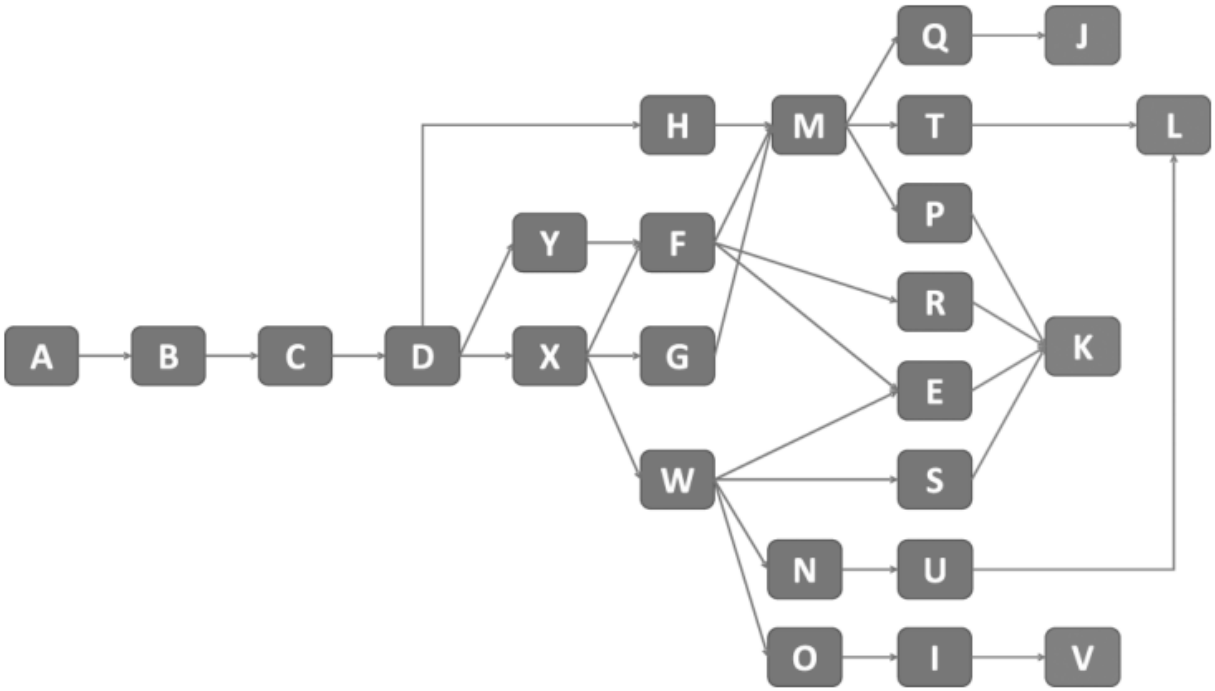


Figure 2.1: A simplified version of the process network used for experiments. The lettered boxes represent processes and the arrows denote that a path uses these processes (from tail to head)

may be split into discrete batches and processed individually (batch splitting) as long as each individual sample passes through the job's specified path in order. To obtain an idea of the type of network we use in this work, we include Figure 2.1 which presents a simplified version of the process network that was used in the experiments throughout this thesis. The network shown was obtained by considering all paths and processes such that the most common 70% of the paths in one month of historical data from a real analytical services facility are considered by this network. We refrain from including the actual network in its entirety because as its size makes it cumbersome to include.

We consider scheduling operations for H minutes from time S until time $S + H$. We refer to the interval $[S, S + H]$ as the time horizon for the problem. Recall that previous works have shown that discrete time representations perform better than continuous time representations for scheduling multipurpose plants [22, 26, 42], therefore we also choose to use a discrete time representation throughout this thesis. Each process $u \in V$ has a set of

timepoints $\varepsilon(u) = \{\varepsilon(u,t) : t = 1, \ldots, |\varepsilon(u)|\}$, where $\varepsilon(u,t) \geq \varepsilon(u,t-1) \;\; \forall t = 2, \ldots, |\varepsilon(u)|$.

Since we schedule or reschedule operations over a facility that is already in operation, it is possible that not all processes and samples are available at the beginning of the horizon. Therefore, to account for this we include some additional input parameters that allow samples to arrive throughout the horizon, and process resources to be unavailable at various points during the horizon. To denote both the samples that arrive at the facility *and* samples that have not finished processing and are carried over from a previous horizon, we use $\alpha(i,k,t)$ to represent the number of samples for job i that arrive at the k'th process in the path of i ($\Pi(i,k)$) at the t'th timepoint of process $\Pi(i,k)$ ($\varepsilon(\Pi(i,k),t)$). $Z(u,t)$ denotes the number of resources of process u that have been pre-allocated at time $\varepsilon(u,t)$, e.g. a machine is scheduled to be taken down for maintenance, or a machine is still running a process that started before the beginning of the current scheduling horizon.

The goal is to generate a schedule (i.e. an assignment of samples to process resources and times) such that an objective of the facility is optimized (e.g. maximize throughput or minimize makespan) and the schedule is feasible. Namely, we require that we do not over allocate our process resources, we abide by path sequencing constraints (e.g. a sample cannot begin processing by a resource which it has not yet reached), and that resource capacity constraints are met. Note that the particular objective that is used will change throughout this thesis, so we do not specify what is used at this time.

2.2 Time Layered Graphs

We now present the definition of a time layered graph, which is needed to develop the scheduling model for the problem described in section 2.1. Time layered graphs may be useful when representing problems which have a network of states, and decisions must be made based on spatial locations in the graph and time, e.g. routing material between states and over a time horizon. The use of these graphs typically arises in transportation routing problems such as bus or flight routing [1, 3, 8, 11]. They augment the standard static network flow models by adding an extra dimension (time) allowing flows to change over time. This time component will be used in this work to model the scheduling of when to start batch processes in a multipurpose facility.

We assume that there is an underlying directed network $G = (V, A)$ where our node set V is comprised of a set of "states" (e.g. physical locations such as stations or holding depots, machines or processes), and our directed arc set A indicates how we may transition from state to state. For each state u, there is a discrete set of integer timepoints $\varepsilon(u)$, indicating

at which times we may leave state u. We denote the k'th timepoint of $\varepsilon(u)$ as $\varepsilon(u,k)$. We denote the set of all sets of timepoints as $\varepsilon = \{\varepsilon(u) : u \in V\}$. Furthermore, we denote the *next* timepoint in the timepoint set of state u after or at time t as $n(u,t) = \min\{t' : t' \in \varepsilon(u), t' \geq t\}$ if it exists, otherwise ∞. At each point in time, t, we assume that there is an integer travel time associated with traveling from state u, to state v, denoted $\tau(u,v,t)$. Note that restricting ourselves to integral points in time for timepoints and integer travel times is without loss of generality (as long as the original time data is rational) as we may discretize time as finely as needed.

With respect to any graph $G = (V, A)$, we denote the set of arcs leaving a node $q \in V$ as $\delta^+_G(q)$, and the set of arcs entering q as $\delta^-_G(q)$. Similarly, we denote the set of nodes that can be reached by traversing a single arc from node $q \in V$ as $N^+_G(q) = \{v \in V : \exists(q,v) \in A\}$, and the set of nodes which can reach q by traversing a single edge as $N^-_G(q) = \{v \in V : \exists(v,q) \in A\}$. When the context of which graph we are discussing is clear, we will omit the subscript to make the notation less cumbersome.

To obtain our time layered graph $G^* = (V^*, A^*)$ we perform the following construction. The set of nodes $V^* = \{(u,t) : u \in V, t \in \varepsilon(u)\}$ is the set of all pairs of states and timepoints for each given state. Arcs either start and end at two distinct states, representing the transition from one state to another, or start and end at the same state, representing staying in the same state. We denote the former set of arcs as A^*_L ("leaving arcs"), and use the next timepoint function $n(v,t)$ and travel function $\tau(u,v,t)$ to add arcs from (u,t) to $(v, n(v, t + \tau(u,v,t)))$ for each neighbor $v \in N^+_G(u)$. The latter set of arcs are denoted as A^*_H ("holding arcs") we add arcs from (u,t) to $(u, n(u,t+1))$. The full set of arcs of G^* are $A^* = A^*_L \cup A^*_H$. The precise algorithm used to generate the time layered graph G^* from the underlying network G and set of timepoints ε is given by algorithm Generate Time Layered Graph shown in table 2.1.

As an example, consider the graph $G = (\{A,B,C,D\}, \{(A,B),(A,C),(B,D),(C,D)\})$, with travel function $\tau(A,B,t) = 2, \tau(A,C,t) = \tau(B,D,t) = \tau(C,D,t) = 1\ \forall t$, and timepoint sets $\varepsilon(A) = \{1,2,3\}, \varepsilon(B) = \{1,2,3\}, \varepsilon(C) = \{2,3\}, \varepsilon(D) = \{1,3\}$. Following the construction given by algorithm Generate Time Layered Graph, the graph G along with the resulting time layered graph G^* are shown in Figure 2.2.

In the proceeding discussions throughout this thesis, when we refer to a time layered graph we will assume that we also have access to the underlying graph G, travel function τ and timepoint sets ε. Moreover, we will use G to denote a graph with no time components and G^* to denote a time layered graph obtained from G (with travel function τ and timepoint sets ε).

Table 2.1: Algorithm 1, Generate Time Layered Graph

Algorithm 1 Generate Time Layered Graph

function BUILD GRAPH(G, ε, τ)
    $V^* \leftarrow$ BUILD NODE SET(G, ε)
    $A^* \leftarrow$ BUILD ARC SET(G, ε, τ)
    return $G^* = (V^*, A^*)$
end function
function BUILD NODE SET(G, ε)
    $V^* \leftarrow \emptyset$
    for all $u \in V$ **do**
        for all $t \in \varepsilon(u)$ **do**
            $V^* \leftarrow V^* \cup \{(u, t)\}$
        end for
    end for
    return V^*
end function
function BUILD ARC SET(G, ε, τ)
    $A^*_L, A^*_H \leftarrow \emptyset$
    for all $u \in V$ **do**
        for all $t \in \varepsilon(u)$ **do**
            if $n(u, t+1) < \infty$ **then**
                $A^*_H \leftarrow A^* \cup ((u, t), (u, n(u, t+1)))$ ▷ Staying at state u
            end if
            for all $v \in N^+(u)$ **do**
                if $n(v, t + \tau(u, v, t)) < \infty$ **then**
                    $A^*_L \leftarrow A^* \cup \{((u, t), (v, n(v, t + \tau(u, v, t))))\}$ ▷ Going from u to v at time t
                end if
            end for
        end for
    end for
    return $A^*_L \cup A^*_H$
end function

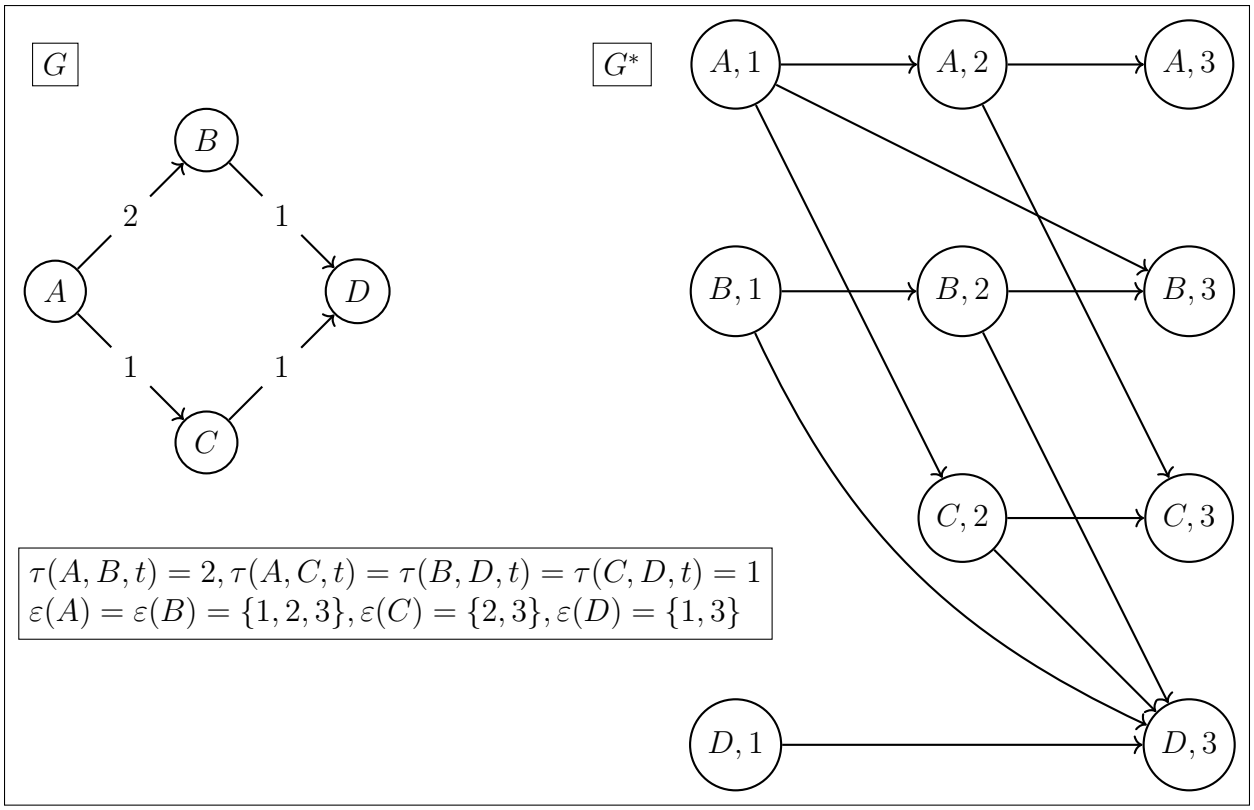


Figure 2.2: An example of how an underlying graph G with travel function τ, and timepoint sets ε produces the time layered graph G^*.

2.3 Scheduling Model

Using the notion of time layered graphs from section 2.2 and the problem description given in section 2.1, we may proceed to present the complete model formulation below. Note that the names used for input data when describing the problem previously will be reused below.

Begin by constructing the graph $G = (V, A)$ as follows. Let the set of nodes be the set of processes of the facility, and add an arc to A from process u to process v if and only if there is a job $i \in I$, $k \in \{2, \ldots, |\Pi(i)|\}$ such that $\Pi(i, k-1) = u$, $\Pi(i, k) = v$. Let the set of timepoints ε correspond to those given as input for each process. For defining the travel function τ of the time layered graph, we use the processing times of each process, i.e. $\tau(u, v, t) = \tau(u)\ \forall u \in V, v \in N_G^+(u), t \in \varepsilon(u)$. With this information, we may construct the time layered graph $G^* = (V^*, A^*)$ as described in section 2.2.

For every job i, every process $\Pi(i, k)$ in the path of job i, and timepoint t for process $\Pi(i, k)$ we have an integer decision variable $x(i, k, t)$ which denotes the number of samples of job i that begin processing on process $\Pi(i, k)$ at time $\varepsilon(\Pi(i, k), t)$. Similarly, we let $w(i, k, t)$ be the number of samples of job i that can be processed on process $\Pi(i, k)$ but instead wait at time $\varepsilon(\Pi(i, k), t)$. For every process u and every timepoint $t = 1, \ldots, |\varepsilon(u)|$ we have an integer decision variable $y(u, t)$ which denotes the number of resources of process u that begin processing samples at time $\varepsilon(u, t)$. Note that the x and w variables correspond to flows on the corresponding arcs in G^*.

Two additional sets are needed to complete the model. The first set is $\Theta_1(i, k, t) = \{t' = 1, \ldots, |\varepsilon(\Pi(i, k-1))| : \varepsilon(\Pi(i, k), t-1) < \varepsilon(\Pi(i, k-1), t') + \tau(\Pi(i, k-1)) \leq \varepsilon(\Pi(i, k), t)\}$. $\Theta_1(i, k, t)$ is the set of timepoints t' such that if process $\Pi(i, k-1)$ starts at t', then it will end in the interval $(\varepsilon(\Pi(i, k), t-1), \varepsilon(\Pi(i, k), t)]$. To put this set into the context of time layered graphs, $\Theta_1(i, k, t)$ is the set of timepoints t' such that a variable $x(i, k, t')$ will have t as its head. The second set is $\Theta_2(u, t) = \{t' = 1, \ldots, |\varepsilon(u)| : \varepsilon(u, t) < \varepsilon(u, t') + \tau(u) \leq \varepsilon(u, t) + \tau(u)\}$. $\Theta_2(u, t)$ is the set of timepoints t' for process u such that if process u started processing at t', then the process would still be running at time $\varepsilon(u, t)$.

We now show the complete model formulation, given by problem $(P1)$. The model takes the form of a *Flexible Discrete-Time Formulation*; for a more in depth discussion of the model along with computational experiments comparing its performance to the continuous

time version, we refer the reader to the original work it appeared in [22].

$$\max_{x,w,y} \sum_{i\in I}\sum_{k=1}^{|\Pi(i)|}\sum_{t=1}^{|\varepsilon(\Pi(i,k))|} f(i,k,t)x(i,k,t) - \sum_{u\in V}\sum_{t=1}^{|\varepsilon(u)|} c(u,t)y(u,t) \tag{$P1$}$$

$$\text{s.t.}\quad x(i,k,t) + w(i,k,t) - w(i,k,t-1) - \sum_{t'\in\Theta_1(i,k,t)} x(i,k-1,t') = \alpha(i,k,t), \quad \forall i\in I, k=2,\dots,|\Pi(i)|, \; t=2,\dots,|\varepsilon(\Pi(i,k))| \tag{1}$$

$$x(i,1,t) + w(i,1,t) - w(i,1,t-1) = \alpha(i,1,t), \quad \forall i\in I, t=2,\dots,|\varepsilon(\Pi(i,1))| \tag{2}$$

$$x(i,k,1) = 0, \quad \forall i\in I, k=1,\dots,|\Pi(i)| \tag{3}$$

$$w(i,k,1) = \alpha(i,k,1), \quad \forall i\in I, k=1,\dots,|\Pi(i)| \tag{4}$$

$$\sum_{i\in I,k'=1,\dots,|\Pi(i)|:\Pi(i,k')=u} x(i,k',t) \le \kappa(u)y(u,t), \quad \forall u\in V, t=1,\dots,|\varepsilon(u)| \tag{5}$$

$$\sum_{t'\in\Theta_2(u,t)} y(u,t') \le \rho(u) - Z(u,t), \quad \forall u\in V, t=1,\dots,|\varepsilon(u)| \tag{6}$$

$$x(i,k,t), w(i,k,t) \ge 0, \quad \forall i\in I, k=1,\dots,|\Pi(i)|, \; t=1,\dots,|\varepsilon(\Pi(i,k))| \tag{7}$$

$$y(u,t) \ge 0, \quad \forall u\in V, t=1,\dots,|\varepsilon(u)| \tag{8}$$

$$x,w,y \text{ integral}, \tag{9}$$

The constraints given by (1) and (2) above are flow conservation constraints on each node of the time layered graph. They ensure that at every point in time for each process in the path of each job (i.e. for every node in the time layered graph), every sample that is available at this point either waits at the current process or is processed. Note that for constraints (1) we have only a single arc leaving the vertex since we have flow constraints for each job i and therefore there is only one arc of A_L^* which may be used (due to path sequencing constraints). Constraints (3) ensure that at the first timepoint, no samples are processed, however note that this is without loss of generality as we may define a second timepoint with the same time as the first timepoint. Constraints (4) set the number of samples that are waiting at the beginning of the horizon based on any samples that arrive at the beginning of the horizon, and any samples that were waiting at the end of the previous horizon. Constraints (5) ensure that enough machines are allocated to meet the proposed schedule. Constraints (6) enforce that we do not allocate more machines than we have available. Constraints (7) - (9) enforce that our variables are non-negative, and integral.

The objective function presented in ($P1$) is designed to maximize the number of samples that begin processing on any process in their path during the horizon with a penalty for

starting process resources. This was done by summing over $x(i,k,t)$ for each job i, each process $\Pi(i,k)$ in the path of job i and each timepoint $t \in \varepsilon(\Pi(i,k))$ and subtracting from the objective each time a process u is used at time $t \in \varepsilon(u)$. This was selected to incentivize the processing of samples even if it was not possible to finish processing them during the time horizon H to ensure that progress is made on finishing jobs. $f(i,k,t)$ in $(P1)$ is used to represent the per sample value of processing the k'th process in job i's path at time $\varepsilon(\Pi(i,k),t)$ and $c(u,t)$ is the per machine cost of starting process u at time $\varepsilon(u,t)$. The specific choices of f and c vary between the two studies presented in Chapters 3 and 4, so we discuss them in these chapters respectively.

Note that $(P1)$ has been presented elsewhere [22, 30] and in these works the time layered graphs were not explicitly constructed. However, time layered graphs were used implicitly for the development of $(P1)$ in these works.

2.4 When to Reschedule

There are various methods that have been proposed for when to trigger reschedulings. "When-to" rescheduling policies can be categorized as *fixed periodic, variable periodic, event-driven*, and *hybrid* [35, 49]. *Fixed periodic policies* invoke rescheduling at equal time intervals, whereas *variable periodic policies* reschedule not based on the time the last rescheduling was done, but instead based on the qualities of the system. For example, rescheduling may be triggered once the difference between the actual schedule and the planned schedule exceeds some allowed tolerance. *Event-driven policies* instead focus on the disruptive events considered by the model and rescheduling is performed whenever a disruptive event occurs. Furthermore, *hybrid policies* use periodic rescheduling in conjunction with event-driven rescheduling when some key events occur, typically events which will cause a greater disruption to the schedule such as a machine breakdown or a rush order arrival [5, 19]. For more information about process rescheduling, we refer the reader to [49] for a thorough review of the subject.

As mentioned previously in Chapter 1, there have been numerous studies conducted on rescheduling in a range of environments. Despite these efforts, a general agreement on how rescheduling frequency affects performance has not been conclusive in the literature. Some works suggest that both scheduling too frequently and not frequently enough result in decreased performance [15, 19]. On the other hand, other studies suggest that one should reschedule as frequently as possible [28, 31, 37], although relative benefits may drop off after some critical point [5, 34, 35]. Hozak and Hill discussed these differences in [18]. They identified some modelling choices that may help explain why the conclusions have

been mixed, such as how instability is modelled, the assumptions used for demands and productions, and not considering human factors. Irrespective of these differing conclusions, it is clear that the performance of different rescheduling frequencies will vary, and hence studying this tradeoff is of interest from a practical standpoint.

The study we conduct on when to reschedule focuses only on fixed periodic policies. Since the disturbances we consider are not highly disruptive to the plant operation, i.e. new jobs arriving will not make the current schedule infeasible, we do not expect to gain much by triggering immediate reschedules which dissuades us from using an *event-driven* or *hybrid* policy. Moreover, there does not seem to be more or less important times of the day to schedule, so using equally spaced timepoints appears to be reasonable. If more serious disruptions were considered, investigating a more complex rescheduling policy could be beneficial to account for the disruptions and is considered as possible future work.

2.5 Time Grids for Discrete Time Representations

Recall that discrete time representations refer to when a time grid (a set of timepoints) is given as input to the scheduling model dictating at what points in time the model may choose to make scheduling decisions. Moreover, there is a tradeoff between how many timepoints are included in the time grid, the CPU cost of solving the model, and the quality of the solution obtained by solving the model [22]. Therefore, the choice of time grid that is supplied to the model can greatly influence the performance characteristics of solving the problem.

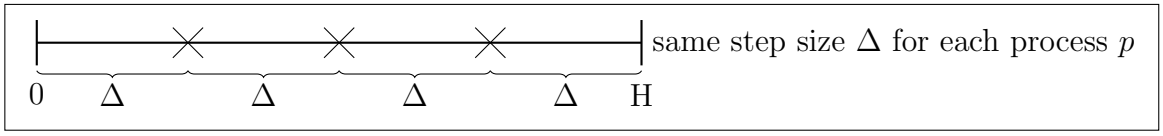
The most common choice of time grid is to use a uniform discretization (UD) with some fixed timestep Δ between points. In fact, until recently, all works on scheduling multipurpose facilities using a discrete time representation did so using a uniform discrete time grid [44]. For uniform discretizations, the timestep may be adjusted to be smaller if more precision in the timepoints is needed, or larger if timepoint precision is less of an issue or CPU solving times are too large. However, the main drawback of this discretization scheme is that the *granularity* of the discretization is constant both over the scheduling horizon and among all processes in the facility. This can be wasteful if one knows beforehand that few or no timepoints are necessary during some interval of time during the scheduling horizon, or if it is known in advance that only a few scheduling decisions will be made for a certain process during the scheduling horizon. Conversely, if one wants to increase the granularity of the time grid during an interval of time, or allow a single process to have more scheduling decisions, then many more timepoints will be added to the time grid than necessary.

A solution to this problem is to consider non uniform discrete (NUD) time grids. NUD time grids assign a (possibly different) set of timepoints for each process such that scheduling decisions for a process p may only occur at timepoints on the time grid of process p [44]. This generalization allows us to overcome the issues discussed previously by adjusting the granularity of the time grid for each process at various points throughout the scheduling horizon. By more carefully selecting where to include timepoints, one may obtain a better tradeoff between CPU time and solution quality than that obtained by using a uniform discretization [22, 44]. Velez and Maravelias [45] showed that using the multi-grid model with their choice of timepoints was guaranteed to produce the optimal solution obtained by using a sufficiently fine global uniform discretization. It has also been shown that using multiple time grids can be suitable for scheduling large scale applications [22, 46].

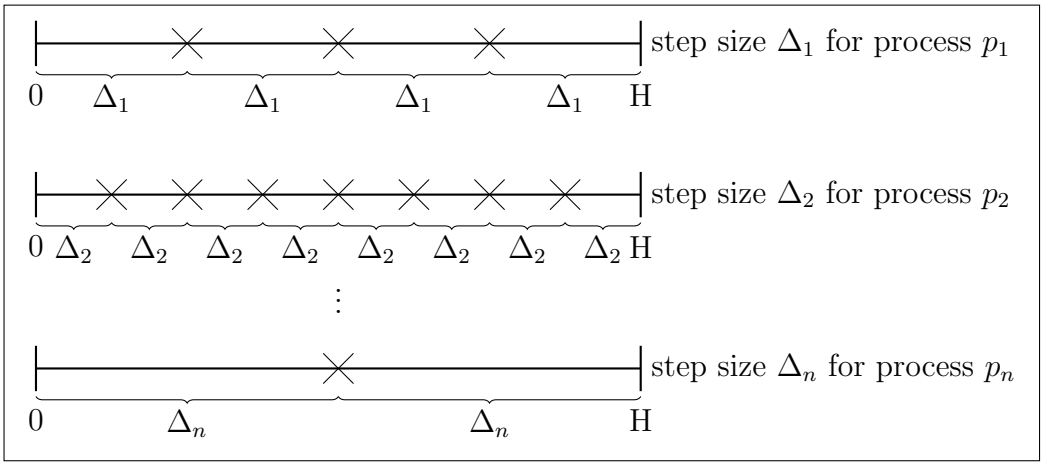
Figure 2.3 demonstrates the difference between uniform discretizations, non uniform discretizations using constant timesteps, and non uniform discretizations using arbitrary timepoints over a time horizon $[0, H]$ where each x marks indicate the presence of a timepoint.

However, guaranteeing that the optimal solution (solution obtained by using an arbitrarily fine time grid) is not cutoff by using multiple time grids may still result in the use of many timepoints and therefore large problems. The non uniform discrete M (NUDM) discretization is one such discretization which does not guarantee that the returned solution is optimal. The NUDM discretization uses the minimum of M and the processing time of a process as the timestep for each process' time grid. This discretization aims to provide timepoints that are not too far apart for processes which have long processing times by selecting an appropriately sized M, and to allow processes with short processing times to begin processing again immediately after finishing a previous run. For the same problem considered in this thesis, the NUD60 discretization ($M = 60$ minutes) was shown in [22] to have a better tradeoff between schedule performance and CPU solving time than the other uniform discretizations that were tested. Therefore, we will refer back to this discretization several times throughout Chapters 3 and 4. Nevertheless, using a NUDM discretization may still suffer from the problem of having many extra timepoints where they are not needed, especially if the considered time horizon is long, or there are many processes with short processing times.

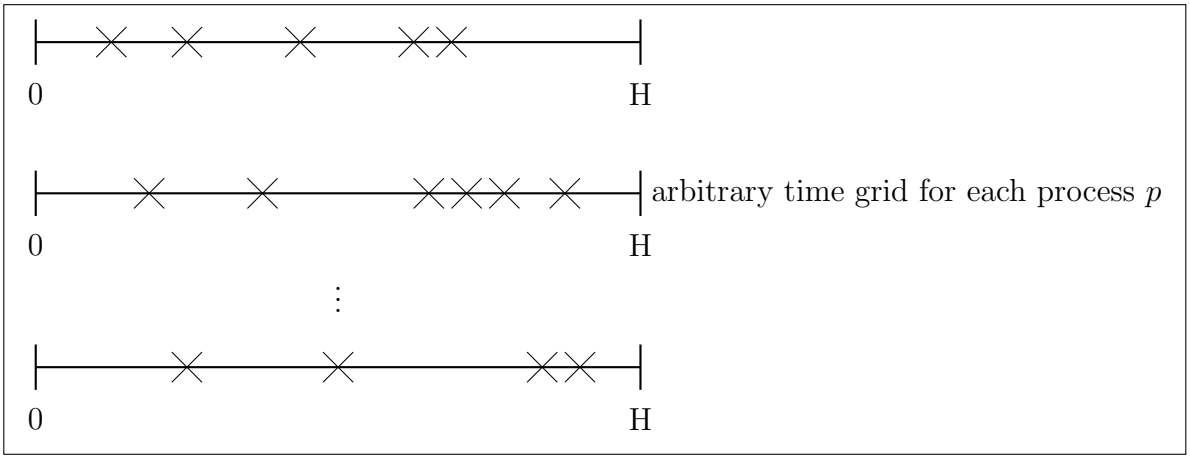
Despite the positive results surrounding using multiple discrete time grids, the pool of works using this method is still very shallow [3, 22, 33, 44, 45, 46]. As alluded to before, choosing a suitable discretization can depend on factors such as the properties of the facility and the length of the scheduling horizon. However, most of these works determine the discretization for their problem either experimentally or by using a time grid which ensures that the optimal solution is not cut off and fix this. The work of Boland et al.



(a) A uniform discretization, each time grid is the same and timepoints are evenly spaced throughout.



(b) A non uniform discretization, each time grid is evenly spaced, however spacing varies between grids.



(c) A non uniform discretization, each time grid has arbitrarily spaced timepoints.

Figure 2.3: Various timepoint discretizations that may be used.

[3] is an exception which does not use a fixed discretization chosen beforehand. Boland et al. [3] employed an iterative method to refine their discretization for the continuous time service network design problem (CTSNDP). They begin by constructing a relaxation to CTSNDP using a carefully chosen time layered graph whose arcs are shorter than the actual travel times for their problem. After solving their relaxation, either the solution may be converted to a solution for CTSNDP, in which case the solution is optimal, or they identify at least one arc to lengthen in their relaxation and repeat this process. They were able to prove that their method terminates with the optimal solution obtained by using a continuous time representation and showed positive results using computational experiments. However, part of their proof of optimality for their method uses a fact which does not hold true in our setting. We attempted to modify their method and proof to fit our setting, but were unable to do so. Nonetheless, their method provided inspiration for the computational work presented in Chapter 4 (even though there is no proof that it will result in the optimal schedule). This work aims to propose a new way of refining the time grids for scheduling problems iteratively, contributing to the lack of such methods using non uniform discrete time representations in the literature.

Chapter 3

Study on Rescheduling Frequency

In this chapter, we perform an investigation on the effects plant parameters play on determining when to reschedule operations. As discussed in Chapter 1 and section 2.4, understanding how often one should reschedule under various facility operational conditions has practical importance and has not been well studied. To study the performance of different rescheduling policies on a facility that receives new job orders throughout the day, we must compare the long term performance of the facility as short term performance differences may be negligible. To address this issue, we use the rolling horizon framework which we present in section 3.1. This framework is used to breakup a horizon that may be impractical to schedule over because of its length into several manageably sized sub-horizons that can be scheduled over in a reasonable amount of time. In section 3.2 we present the various rescheduling policies that were tested in our experiments. Section 3.3 details the various metrics which were used to evaluate a policy's performance. In section 3.4 we discuss the model parameters that were used for our experiments and the design of our experiments. Section 3.5 presents the results of the experiments and a discussion on the results, leading to some recommendations on choosing a suitable rescheduling policy based on the properties of the production environment. A summary of the chapter is given in section 3.6.

We want to note that the work in this chapter was submitted for publication. Z. Stevenson, R. Fukasawa, L. Ricardez-Sandoval, "Evaluating periodic rescheduling policies using a rolling horizon framework in an industrial-scale multipurpose plant". This manuscript was co-authored by myself, and my supervisors, Dr. Fukasawa and Dr. Ricardez-Sandoval and was submitted to the Journal of Scheduling published by Springer on October 25, 2018.

3.1 Rolling Horizon Routine

A rolling horizon routine was used with the model ($P1$) presented in section 2.3 to simulate the operation of the facility over long periods of time (e.g. months). Decomposing the scheduling horizon into several smaller, contiguous time horizons is necessary as scheduling over the entire horizon at once would be computationally intractable. Furthermore, when scheduling operations in practice for a facility whose future arrivals are not known in advance, and over an indeterminate amount of time, the operator must implicitly use a rolling horizon strategy to schedule operations. The choice of how often to reschedule operations and how long the horizons should be will impact when new job arrivals are considered by the model, the computational cost of scheduling, and the quality of the actual implemented schedule. For these reasons, it is important to gain insight on the tradeoffs of using different policies so that an operator can make an informed decision when choosing how to reschedule operations in practice.

The routine can be thought of as partitioning the entire horizon that needs to be scheduled into smaller sub-horizons and then solving each of these sub-horizons sequentially. Let Γ denote the time horizon that needs to be scheduled, and denote the i'th sub-horizon by Γ_i for some partitioning of Γ into n sub-horizons, see figure 3.1. We start by setting $i = 1$ and the state of the facility as having an initial set of jobs arriving at time S and all machines empty. We then solve the model ($P1$) over the sub-horizon Γ_i to generate a schedule $Sched_i$. Denote the set of jobs that arrive at the facility during sub-horizon Γ_i as α_i. The state of the facility is updated using $Sched_i$ and α_i to reflect the current jobs and the current machine usage at the beginning of sub-horizon Γ_{i+1}. After updating the state of the facility, we then check if there are other sub-horizons to schedule over. If there are, then we increment i and repeat the process. If we have scheduled over all of the sub-horizons, then we stop and concatenating ($Sched_1$, ..., $Sched_n$) gives us a feasible schedule for the entire horizon Γ. Note that this process involves creating and solving a model ($P1$) for each sub-horizon to obtain the corresponding schedule.

After generating a schedule $Sched_i$ and considering arrivals α_i for sub-horizon Γ_i, several operations must be performed to update the state of the facility for the following sub-horizon. These operations are done to maintain consistency of the facility between horizons and we describe them next. Jobs which arrive during the sub-horizon Γ_i have their arrival times pushed back to the start of sub-horizon Γ_{i+1} since we may not consider these jobs during the current horizon but want the model to be able to schedule them at the beginning of the next sub-horizon. Note that we push these samples back because we schedule each sub-horizon once using the information available at the beginning of the sub-horizon. Samples which arrived at the facility prior to sub-horizon Γ_i are handled according to

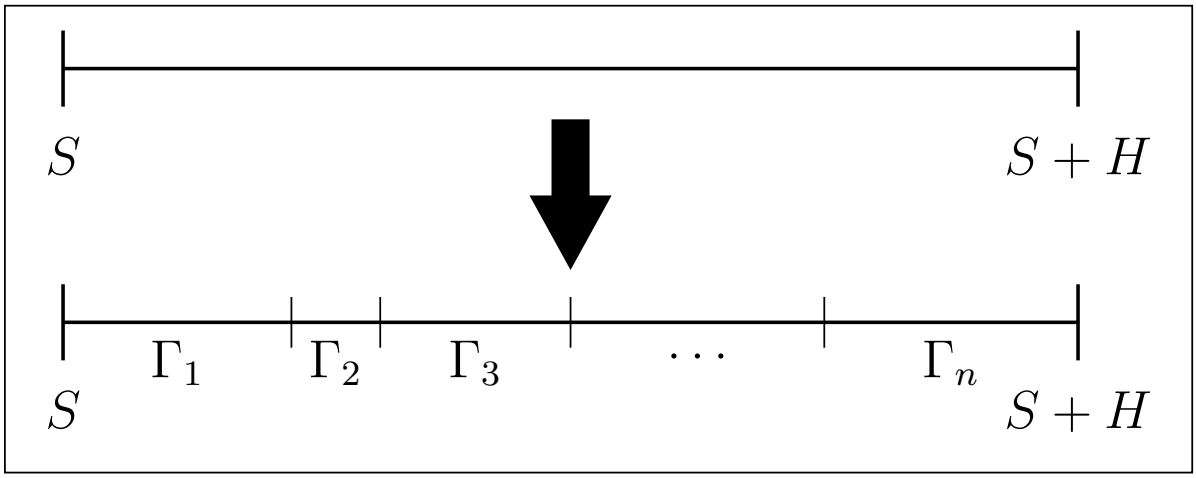


Figure 3.1: Above is the scheduling horizon before partitioning. Below is a partition of scheduling sub-horizons that we may actually solve.

the following rules. Consider a variable $x(i', k, t) > 0$ which is scheduled during Γ_i. If this variable corresponds to samples that are actively being processed at the end of Γ_i (i.e. $\varepsilon(\Pi(i', k), t) + \tau(\Pi(i', k)) \in \Gamma_{i+1}$), then a corresponding arrival $(\alpha(i', k+1, t'))$ for these samples is created in Γ_{i+1} for the next process in their path, $\Pi(i', k+1)$, at time $\varepsilon(\Pi(i', k), t) + \tau(\Pi(i', k))$ (e.g. when the samples will finish being processed by $\Pi(i', k)$). Similarly, consider a variable $w(i', k, t) > 0$ which is scheduled during Γ_i, such that t is the last timepoint of process $\Pi(i', k)$. In this case, a corresponding arrival $(\alpha(i', k, t'))$ is created in Γ_{i+1} for the same process they were waiting for, $\Pi(i', k)$, at the beginning of Γ_{i+1}. Note that these two cases cover all of the samples scheduled during Γ_i as each sample is either being processed at the end of the horizon, or is waiting to be processed. Any process resources that are running at the end of Γ_i (i.e. $y(u, t) > 0, \varepsilon(u, t) + \tau(u) \in \Gamma_{i+1}$), are accounted for by adding $y(u, t)$ to the $Z(u, t')$ variables in Γ_{i+1} for all timepoints $t' \in \varepsilon(u)$ such that $t' < \varepsilon(u, t) + \tau(u)$.

This procedure allows us to generate a schedule which spans Γ by solving smaller sub-problems when Γ is prohibitively large to schedule all at once. However, it is worth noting that even without considering new arrivals, this procedure is a heuristic with respect to scheduling over the entire horizon. Each sub-schedule $Sched_i$ may be optimal with respect to its corresponding sub-horizon, but the concatenated schedule ($Sched_1$, ..., $Sched_n$) will not necessarily be optimal with respect to Γ due to the myopic nature of scheduling over each sub-horizon individually.

Table 3.1: Descriptions of rescheduling policies considered.

ID	Description
4P	Schedule the current day's operations at the beginning of every day and revise the remainder of the day's schedule 3 times during the day at equal intervals
2P	Schedule the current day's operations at the beginning of every day, then revise the second half of the schedule halfway through the day
S	Schedule the current day's operations at the beginning of every day
3D	Schedule the next three days of plant operation at the beginning of every third day
5D	Schedule the next five days of plant operation at the beginning of every fifth day

3.2 Rescheduling Policies

We now describe the rescheduling policies that were considered in this study. Recall that we consider only fixed periodic rescheduling policies during this study for the reasons discussed in section 2.4. Table 3.1 gives a summary of the rescheduling policies considered in this work.

As shown in Table 3.1, policies "4P", and "2P", which can be thought of as "4 Parts", and "2 Parts", both schedule operations more than once per day at equal intervals. For example, if we consider scheduling operations for twenty four hours per day with the "4P" policy, then we will first generate a schedule for the next twenty four hours at the beginning of the day. We will follow this schedule for the first six hours, and then a new schedule will be generated for the following eighteen hours of the day. We will then generate a schedule for the remaining twelve hours after twelve hours and similarly for the last six hours after eighteen hours. These rescheduling policies emulate the operation of a facility that is concerned with the short term, day to day operations and wishes to reschedule frequently based on new job arrivals. The reason we do these reschedulings is because any new jobs that arrive during the first six hours may not be scheduled at the beginning of the day as they have not yet arrived at the facility, however by rescheduling after six hours, we may generate a new schedule for the remainder of the day that considers these new arrivals. The "2P" policy uses a similar strategy, but only schedules operations twice, once at the start of the day and again halfway through the day.

The "S" policy, referred to as "Single", generates a single schedule for the current day at the beginning of the day and will follow it, ignoring new job arrivals until a new schedule is generated at the beginning of the following day. This policy simulates a facility that is concerned with day to day operations but will not disrupt or modify the current schedule. The "3D", and "5D" policies which can be thought of as "3 Days", and "5 Days", schedule operations for the next three and five days, respectively. During this time, these policies behave like the "S" policy, ignoring new job arrivals until the next time operations are rescheduled. For this reason, in the worst case scenario when scheduling with the "5D" policy, the model may not be aware of a job which arrives at the facility until five days after it initially arrived. These policies simulate a facility that is more concerned with schedule stability, rather than reacting quickly based on the newest job arrivals.

These period lengths were chosen for testing so that we may test a wide range of policies and also policies which may be typically used in practice. For instance, the "5D" policy which corresponds to scheduling on a weekly basis, and the "S" policy which corresponds to scheduling once per day are both natural candidates for rescheduling. The "4P" policy was chosen to obtain information about how rescheduling very frequently would perform and the "2P" and "3D" policies were chosen to test policies which fell between the "S" policy and the other two extremes.

These policies may also be related to the rolling horizon framework discussed in section 3.1. The "5D" policy corresponds to using sub-horizon lengths of 120 hours for each Γ_i. Similarly, the "3D" and "S" policies correspond to sub-horizon lengths of 72 hours and 24 hours respectively. The "2P" and "4P" policies differ however, as discussed above, these policies schedule the remainder of the current day at equal intervals. Therefore, the "4P" policy corresponds to using sub-horizons of length 24, 18, 12, and 6 hours to schedule each day. The "2P" policy operates similarly with sub-horizons of length 24 hours and 12 hours each day.

3.3 Performance Metrics

To compare the performance of the various policies considered in this study, we use job completion, average job makespan, and proportion of jobs on time. We begin by presenting the formulation of these metrics, before discussing them below.

Let $arr(i)$ be the arrival time of job i, $fin(i)$ be the time that the last sample of job i finishes processing, $jobs(t) = \{i : arr(i) \leq t\}$ be the set of jobs that arrive before or at time t, and $comp(i,t) = 1$ if job i has finished processing before or at time t, and 0 otherwise.

Then job completion at time t, average job makespan at time t, and proportion of jobs on time at time t with lead times of d minutes were defined as follows:

$$completion(t) = \frac{\sum_{i \in jobs(t)} comp(i,t)}{|jobs(t)|} \tag{3.1}$$

$$avg\_makespan(t) = \frac{\sum_{i \in jobs(t):comp(i,t)=1} fin(i) - arr(i)}{|\{i \in jobs(t) : comp(i,t) = 1\}|} \tag{3.2}$$

$$on\_time(d,t) = \frac{|\{i \in jobs(t) : comp(i,t) = 1, fin(i) - arr(i) \leq d\}|}{|jobs(t)|} \tag{3.3}$$

For comparing policies using these metrics, we use the value measured at the end of the final scheduling horizon. Since we do not require that all jobs finish processing during the corresponding scheduling horizon, for measuring average job makespan we consider only those jobs that have finished processing at the end of the time horizon. Job completion at time t was defined as the proportion of jobs that finished processing all samples out of all the jobs that have arrived at the facility between the beginning of the first horizon and t. Makespan was taken to be the difference in minutes between when the final sample finished processing for a given job and when that job initially arrived at the plant, and was only considered for jobs that have finished processing. Proportion of jobs on time with respect to some lead time d was defined as the number of jobs that finished within the lead time d out of all jobs that had arrived before time t.

3.4 Design of Experiments

In this section we go over the details relating to how the experiments were carried out for this study. We begin by describing the facility that was considered in more detail. As discussed in section 2.3, the plant used for our experiments is based on a multipurpose industrial-scale analytical services facility. Due to confidentiality agreements, we cannot disclose detailed data. The facility is rather large with nearly 200 distinct processes, each of which may have multiple identical machines. During a thirty day timespan, the facility received jobs comprising of over 150 unique paths, using approximately 100 unique processes. Over this timespan, they received several hundred jobs comprising of more than 20,000 samples. This large volume of jobs leads to large formulations when using long horizon lengths in the rolling horizon routine described in section 3.1. The capacities and processing times of the individual processes vary greatly. The largest capacity among all

processes is over 1,300 times the size of the smallest capacity, similarly the processing times of the processes vary from a few minutes to several days. In Appendix A, we present normalized values for capacity, processing time, and number of resources for each process. These qualities differentiate the plant studied from the simpler and smaller facilities found in other rescheduling studies [15, 37, 48]. Although our experiments use a plant based on the facility of our industrial partner, the model defined in Section 2.3 is a general job shop formulation that may be used for other industrial applications that require a multipurpose plant. Therefore, the same methodology and experiments may be performed using other applications of multipurpose plants.

The following choices for parameter values for the model $(P1)$ were used throughout the experiments. The time grid given to the model, ε, was chosen to be the NUD60 discretization used in [22] as it was shown to have a good tradeoff between solving time and solution quality for this problem, compared to the other discretizations tested. Recall that for the NUD60 time discretization, each process u has an associated timestep $\Delta(u) = \min\{60, \tau(u)\}$ and the timepoints for process u are $\varepsilon(u) = \{S, S+\Delta(u), \ldots, S+(\lfloor H/\Delta(u)\rfloor)\Delta(u), S+H\}$. This choice of timepoints allows processes which have short processing times to have fine granularity, and also allows us to be flexible with the scheduling of long processes by including a timepoint each hour.

We will now discuss the choice of objective function used for our experiments. We compare the performance of the various policies based on job throughput, average job makespan, and proportion of jobs on time, over the scheduling horizon as described in section 3.3. However, we do not optimize these metrics directly because of the inherent difficulties associated both with optimizing multiple and conflicting objectives, and the length of some job paths being longer than the lengths of the subhorizons used. For instance, suppose job throughput was our main objective and hence the objective function only provided incentive to processing the last process in each job's path. If we consider a job whose path length is longer than the length of the subhorizon, then the model would have no incentive for processing the job through earlier processes in its path and potentially no progress would be made on the job. Instead of optimizing these metrics directly, we chose the values for the objective function parameters of model $(P1)$ to be the following:

$$\begin{aligned} f(i,k,t) &= \left(1+\tfrac{|\varepsilon(\Pi(i,k))|-t}{|\varepsilon(\Pi(i,k))|}\right)\left(\tfrac{k}{\sum_{m=1}^{|\Pi(i)|} m}\right), \\ &\quad \forall i \in I, k = 1,\ldots,|\Pi(i)|, t = 1,\ldots,|\varepsilon(\Pi(i,k))| \\ c(u,t) &= 0.001,\ \forall u \in V, t = 1,\ldots,|\varepsilon(u)| \end{aligned}$$

These values were chosen to put a larger weight on processing samples that are further along in their path, and also to put a higher priority on processing samples earlier in

the horizon. The rationale behind these decisions was that, by prioritizing samples that were closer to being finished we would push currently open jobs toward completion before starting new jobs. Furthermore, if a schedule could be shifted in time we would prefer it be executed as early as possible so that resources may be left unused to accommodate for possible job arrivals in the future. This was accomplished by assigning more weight to processing samples earlier in the horizon. We also assign a cost for using resources so that optimal schedules will allocate process resources if and only if the schedule assigns samples to be run on those resources. These decisions were done in an attempt to obtain attractive solutions (measured in terms of job throughput, job makespan, and proportion of jobs on time), in the rolling horizon framework. We note that these choices for the objective function parameters do not assign a cost to schedule disruptions between horizons. These costs are more difficult to quantify and are beyond the scope of this study.

We now describe the experimental procedure that was used. We let the initial state of the facility be empty with no machines running ($Z(u,t) = 0,\ \forall u,t$) and an initial influx of jobs all arriving at the beginning of the horizon (time S). We carry out the rolling horizon routine from section 3.1 for 60 days, assuming that new jobs arrive uniformly at random throughout each day. We decided to carry out the experiments for 60 days so that the plant was able to reach a stable operation and then continue running to obtain results when measuring proportion of jobs on time with one month lead times.

Unless otherwise specified, the following methods were used for the experiments presented in section 3.5. To assign job paths, actual job arrivals to the facility were recorded over the span of a high production month using historical data from the plant. Job paths were sampled according to the observed frequencies of each path during this timespan. The number of samples in each job was selected uniformly at random between ten and fifty, which was determined based on the observed historical plant data during this month.

We allowed the facility to be fully operational for the first eight hours each day. During this period, the plant operates with complete staffing and all available resources may be used to process samples. During the following sixteen hours each day we did not allow new operations to begin, but previously started operations were allowed to continue processing. The way this was handled with the "2P" policy was to first schedule the eight hour shift, and then revise the schedule for hours five through eight after hour four. Similarly, we revise the remainder of the schedule after hours two, four, and six for the "4P" policy. This was done to simulate actual plant operation. Workers may begin new operations while they are on site during the first eight hours each day, during the following sixteen hours only a few workers are present to supervise the operation of previously started processes. After fixing the design parameters for each experiment, ten random instances with different job arrivals and starting jobs were generated and scheduled over to obtain an accurate representation

of the results.

3.5 Results

The present study was performed on a 48 core machine running at 2.3GHz, with access to 256GB of RAM. The implementation was done using the Julia programming language (version 0.6.2), the CPLEX.jl (version 0.3.1) and JuMP.jl [10] (version 0.18.0) packages, and CPLEX (version 12.6.0.0) [7] for the solver. All of the CPLEX parameters were set to their default values, except for a limitation to use only 2 CPU cores, setting the relative MIP gap to be 0.5%, and imposing time limits on the instances. Time limits were set to be 15 minutes for each day being scheduled in the horizon, e.g. if the instance was using a "3D" policy as described above, then the time limit was set to be 45 minutes. This policy was set by the industrial partner based on their scheduling requirements. The time limits were met by some instances that solved three or five days consecutively, but for all of the problems that reached the time limit the relative optimality gap was at most 5% and was typically less than 1%.

Performance profiles are one of the main representations used below for showing the results of the experiments. In our context, we compare the performance of the different rescheduling policies over a large test set and use performance profiles to show the performance of each policy relative to the best performing policy with respect to some metric over each of the tests in the test set. Examples will be given in the following section as to how to read these figures. For more detailed information about performance profiles we refer the reader to [9].

3.5.1 Results with Moderate Facility Load

The experiments described next were considered to obtain results that were representative of a facility that received enough new jobs to be running constantly, but not enough that the queue of waiting jobs grows indefinitely. We considered the parameters of the problem to be the following: the facility starts with 5,000 samples (approximately 170 jobs) arriving at the beginning of the first horizon. Jobs arrive at the facility throughout each day according to the following sampling method: first an arrival time for the job is selected uniformly at random during the horizon, next a number of samples is selected uniformly at random between ten and fifty, now if the total number of samples set to arrive during the horizon is at least 500, then we stop, otherwise, we sample another arrival time

for another job and repeat. The facility is operated for 8 hours each day as described in section 3.4. These numbers of samples were determined experimentally. The names of the plots in the figures below have the following format: (number of samples added per day) - (rescheduling policy), for instance "500 - 4P" corresponds to the performance of rescheduling four times per day for instances where 500 samples are added each day over the 60 day horizon.

Figure 3.2 shows a performance profile comparing average job completion percentage over the 60 day horizon between the different rescheduling policies. On the x-axis we have the performance of each policy measured as a factor of the performance of the best performing policy on each test. Note that we are comparing against the best performing policy tested, not necessarily the true optimum. On the y-axis we have the proportion of tests that a policy achieves within a given factor of the optimal performing policy for each test. This Figure shows that the "4P" policy clearly dominates the other policies. The point $(1, 1)$ in the curve of the "4P" policy in the figure indicates that it was the best performer in all of the tests. Similarly, the point $(1.02, 0.5)$ in the curve of the "3D" policy indicates that in 50% of the instances, the "3D" policy performed within 2% of the best performing policy for that instance. We also observe that rescheduling more frequently continues to improve job completion performance for each of the policies tested. Furthermore, in the worst case, rescheduling only once every five days is approximately 5.6% worse than rescheduling four times per day. This is indicated by the point $(\sim 1.056, 1)$ in the curve of the "5D" policy and by noting that for all smaller x values, the curve lies below 1. Therefore, depending on how much importance is placed on schedule stability, it may be worth it to delay the rescheduling of operations to only once every few days.

Figure 3.3 shows a performance profile comparing average job makespan performance over the scheduling horizon between the different rescheduling policies. From this figure we draw the same conclusion as for job completion (that more frequent rescheduling is better), however the performance differences between the policies are much larger for this metric. For instance, we observe that in all of the tested instances, using the "5D" policy results in an average job makespan that is over twice as large as the best performing policy. In this case, if job makespan is an important objective of the facility, there is less of a case to be made for delaying the scheduling of new arrivals rather than scheduling them as soon as possible.

Table 3.2 summarizes the results obtained when considering the other metrics. The performance profiles for the other metrics are similar to Figure 3.2, so instead of presenting each performance profile, we present the average performance factor (APF) values as a more informative measure of relative differences. The APF of a policy is defined to be the mean performance (measured as a factor of the best performing policy) a policy achieves over

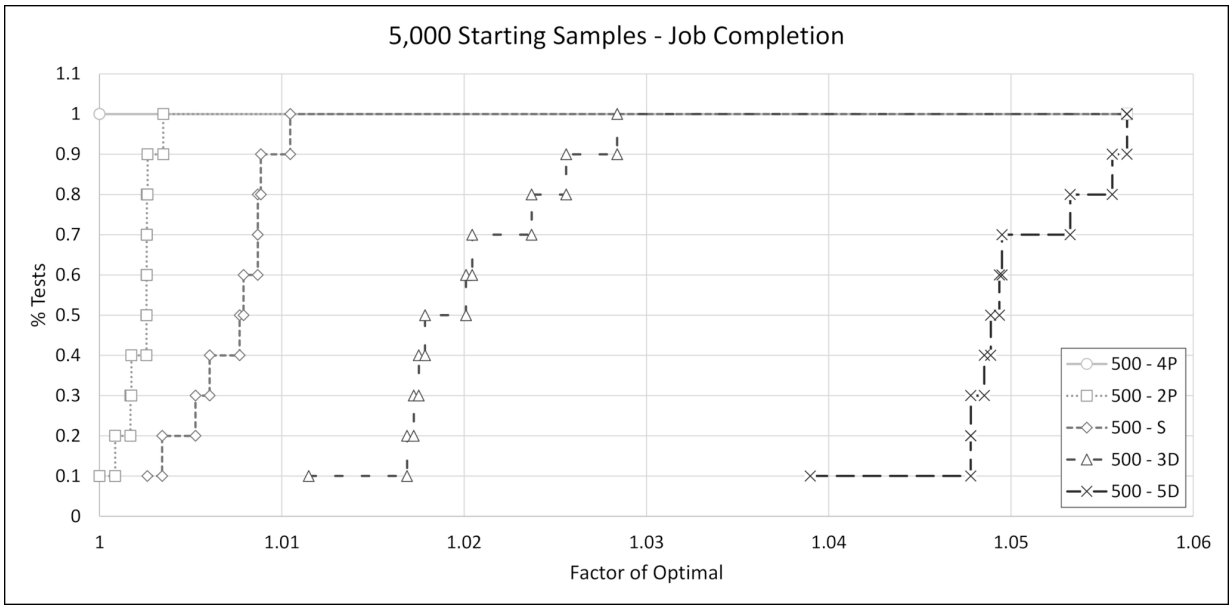


Figure 3.2: Performance profile comparing job completion among rescheduling policies for instances starting with 5,000 samples and 500 samples arriving each day.

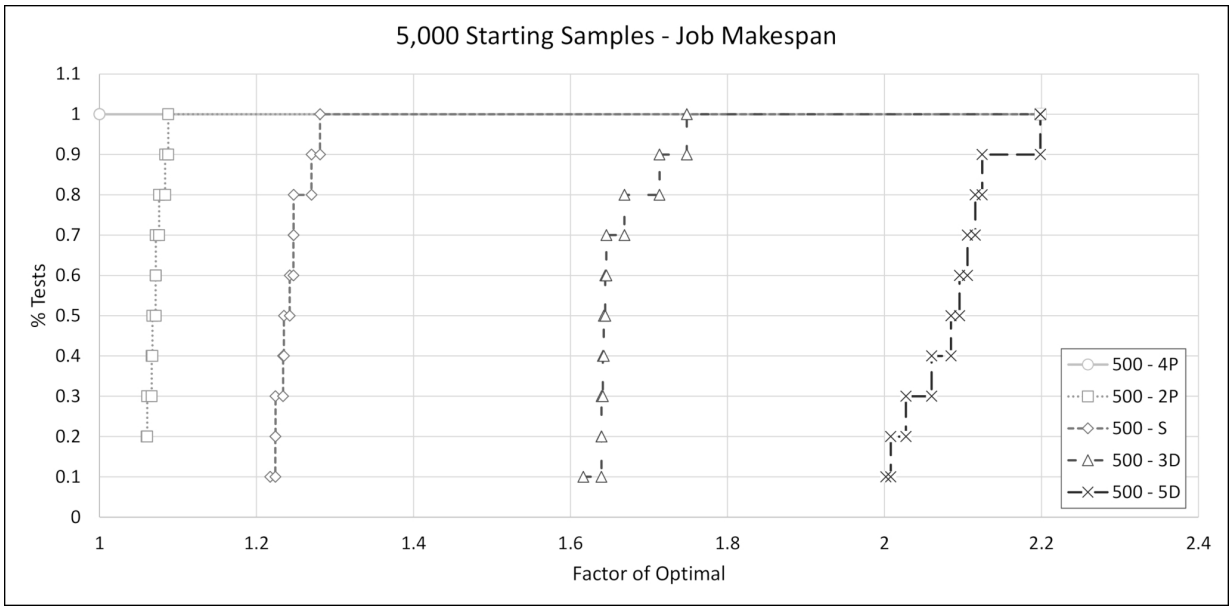


Figure 3.3: Performance profile comparing average job makespan among rescheduling policies for instances starting with 5,000 samples and 500 samples arriving each day.

the ten random instances. The APF was used to compare the relative performance of each policy and estimate the differences in APF values between pairs of policies when decreasing rescheduling frequency from most to least frequent, i.e. when moving from the "4P" policy toward the "5D" policy.

From Table 3.2 we draw the same conclusion as above: rescheduling more frequently improves performance. The degree of performance gain is dependent on the chosen metric, and can vary from negligible improvements to an order of magnitude difference between the "4P" and "5D" policies in some cases. Furthermore, we observe that the relative benefit obtained by moving from less frequent rescheduling policies to more frequent rescheduling policies diminishes as one begins to reschedule frequently. This is demonstrated by the generally increasing APF differences as we move from the "4P" policy toward the "5D" policy. We note that the performance differences between policies also decrease in general with respect to proportion of jobs on time as lead times are increased. Therefore, if longer job turn around times are acceptable, then the smaller performance loss may be a worthwhile tradeoff for the added schedule stability achieved by rescheduling only once every few days. For more detailed results, we include the performance profiles which generate Table 3.2 in the Appendix B.

Table 3.3 shows the average number of variables and constraints that were used when solving each model for the industrial-scale facility. To clarify, these numbers correspond to the average size of each model that was solved for each of the policies tested. Note that each policy may solve a different number of models to generate a schedule over the same scheduling horizon. For instance, the "4P" policy generates a schedule four times per day, and hence solves 240 models over a 60 day horizon, whereas the "5D" policy will solve 12 models over the same 60 day horizon. We also include average problem solving times. As expected, the average time required to solve each model increases as the lengths of the horizons scheduled in each model increases. However, we also note that there is a tradeoff between solving smaller models and the total number of models that are solved over the 60 day horizon by observing the total solving times for the different policies. We observe that scheduling once per day using the "S" policy requires the least amount of time, scheduling multiple times per day increases the total solving time moderately, and scheduling over long horizons for the "3D" and "5D" policies increases the total solving time considerably. This drastic increase in cost when scheduling over multiple days can be attributed to the much larger problems that are being solved. The problems are larger with respect to the number of timepoints considered because of the added length of the horizon. Furthermore, by stockpiling new job arrivals for several days, accounting for all of them during the next horizon increases the number of jobs considered, which further increases the problem size.

Overall, the results obtained by using nominal values for load parameters agree with

Table 3.2: Average performance factor (APF) values of each metric for instances with 5,000 starting samples and 500 arriving samples per day.

ID	Job Completion APF (Difference)
4P	1.0 (-)
2P	1.0021 (0.0021)
S	1.007 (0.0049)
3D	1.0199 (0.0129)
5D	1.0496 (0.0297)
ID	Job Makespan APF (Difference)
4P	1.0 (-)
2P	1.0705 (0.0705)
S	1.2422 (0.1717)
3D	1.66 (0.4178)
5D	2.0822 (0.4222)
ID	Proportion of Jobs On Time APF, 1 Day Lead Times (Difference)
4P	1.0 (-)
2P	1.2256 (0.2256)
S	2.5536 (1.328)
3D	8.1484 (5.5948)
5D	11.059 (2.9106)
ID	Proportion of Jobs On Time APF, 1 Week Lead Times (Difference)
4P	1.0 (-)
2P	1.0031 (0.0031)
S	1.0113 (0.0082)
3D	1.0307 (0.0194)
5D	1.1075 (0.0768)
ID	Proportion of Jobs On Time APF, 1 Month Lead Times (Difference)
4P	1.0002 (-)
2P	1.0021 (0.0019)
S	1.007 (0.0049)
3D	1.0204 (0.0134)
5D	1.05 (0.0296)

Table 3.3: Problem size statistics for instances with a moderate job load.

	4P	2P	S	3D	5D
Average Number of Variables Per Model	56,212	59,297	65,251	234,488	522,046
Average Number of Constraints Per Model	44,525	46,092	49,123	165,283	340,737
Average Solving Time Per Model (s)	0.64	0.96	1.82	16.8	86.5
Total Solving Time for 60 Day Horizon (s)	153	115	109	336	1,038

the most common conclusion in the literature that more frequent rescheduling is better and that the relative benefit may diminish as rescheduling frequency increases. We noted that the degree of this benefit varies depending on the performance measure chosen for comparison.

3.5.2 Effects of Different Plant Loads

To observe the effect of plant load on the results, we generated new instances that varied both the starting number of samples and the number of arriving samples per day from their nominal values. In particular, we kept the number of starting samples constant but varied the number of arriving samples per day in some experiments. In other experiments, we varied the number of starting samples but kept the number of samples arriving per day constant. For each of these pairs of number of starting samples and number of arriving samples per day, we generated and solved ten random instances with 60 day horizons, which were used to produce the results shown in Table 3.4 and Table 3.5.

Table 3.4 shows the APF values for each policy, both in terms of job completion and average makespan, when the number of starting samples was varied, and the number of samples added per day was fixed to 500. We tested with 2,500, 5,000, and 10,000 starting samples as shown in Table 3.4 (column 2). By observing the job completion and average makespan APF differences for the three different starting loads, we note that, as the number of samples available at the beginning of the experiment increases, the performance difference between policies decreases. We also note that more frequent rescheduling again performs better.

Table 3.5 shows the APF values for each policy when the number of starting samples

is fixed to 5,000 and the number of samples arriving daily is varied. Similar to above, we tested with 250, 500, and 1,000 samples arriving per day and calculated the APF differences between policies as rescheduling frequency is decreased, for these different loads. Contrary to the observations for Table 3.4, Table 3.5 shows that increasing the number of daily sample arrivals actually increases the performance differences observed between policies. However, we again note that more frequent rescheduling appears to perform better.

Note that by increasing the amount of samples arriving at the beginning of the horizon, the plant begins the experiment with a larger queue of jobs. As a consequence of this, all policies have a *backlog* of jobs to schedule and hence there is less advantage to receiving new job arrivals more promptly, since there are already many jobs to schedule. It is sensible then that the performance differences between policies decreases as the number of starting samples increases. Similarly, when more samples are received at the facility on a daily basis, by rescheduling frequently the model is able to consider these many new samples earlier than policies which will not consider these samples until at least the next day. Therefore, it is reasonable that performance differences increase as the number of daily sample arrivals increases. We also note that these results suggest that plant capacity may play a role, as we observe that the differences between policies is variable with respect to the load on the facility and the presence of a backlog of jobs.

With regards to how the proportion of jobs on time reacts to different loads on the facility, we largely observe the same trends as when moderate loads were used above. That is, as the number of starting samples was increased, the performance differences between policies decreased. However, as the number of daily job arrivals was increased, the performance differences between policies increased as well. These observations agree with the results discussed above concerning makespan and job completion, and follow from the same discussion. We omit the inclusion of these Figures here for brevity, but include them in the Appendix B.

3.5.3 Effects of Varying Plant Capacity

Based on the observations obtained from the previous scenario, we investigate the role plant capacity plays on the rescheduling frequency. In the following experiments, the capacity of all the processes in the plant was tested at 8x, 4x, 2x, $\frac{1}{2}$x, $\frac{1}{4}$x, and $\frac{1}{8}$x their original capacity. These values were chosen to obtain data for a wide range of overall capacities with the $\frac{1}{8}$x representing a facility that is severely starved for resources and the 8x representing a facility which has more than enough resources to process the given demand. We performed ten random instances for each plant capacity, and instances were run with moderate job

Table 3.4: APF differences comparison for instances starting with various numbers of samples and 500 new samples per day.

ID	Starting Samples	Job Completion APF	Job Completion APF Differences	Average Makespan APF	Average Makespan APF Differences
4P	2,500	1.0	-	1.0	-
2P	2,500	1.002	0.002	1.0658	0.0658
S	2,500	1.0083	0.0063	1.2333	0.1675
3D	2,500	1.0213	0.013	1.6709	0.4376
5D	2,500	1.0537	0.0324	2.1863	0.5154
4P	5,000	1.0	-	1.0	-
2P	5,000	1.0021	0.0021	1.0705	0.0705
S	5,000	1.007	0.0049	1.2422	0.1717
3D	5,000	1.0199	0.0129	1.66	0.4178
5D	5,000	1.0496	0.0297	2.0822	0.4222
4P	10,000	1.0	-	1.0	-
2P	10,000	1.0019	0.0019	1.0559	0.0559
S	10,000	1.0068	0.0049	1.1707	0.1148
3D	10,000	1.0179	0.0111	1.415	0.2443
5D	10,000	1.0418	0.0239	1.7077	0.2927

Table 3.5: APF differences comparison for instances starting with 5,000 samples and various daily arrival loads.

ID	Samples Added Per Day	Job Completion APF	Job Completion APF Differences	Average Makespan APF	Average Makespan APF Differences
4P	250	1.0001	-	1.0	-
2P	250	1.0024	0.0023	1.0657	0.0657
S	250	1.0083	0.0059	1.2145	0.1488
3D	250	1.0221	0.0138	1.5785	0.364
5D	250	1.0477	0.0256	1.9481	0.3696
4P	500	1.0	-	1.0	-
2P	500	1.0021	0.0021	1.0705	0.0705
S	500	1.007	0.0049	1.2422	0.1717
3D	500	1.0199	0.0129	1.66	0.4178
5D	500	1.0496	0.0297	2.0822	0.4222
4P	1,000	1.0002	-	1.0	-
2P	1,000	1.0029	0.0027	1.0658	0.0658
S	1,000	1.0099	0.007	1.2333	0.1675
3D	1,000	1.0263	0.0164	1.6709	0.4376
5D	1,000	1.0556	0.0293	2.1863	0.5154

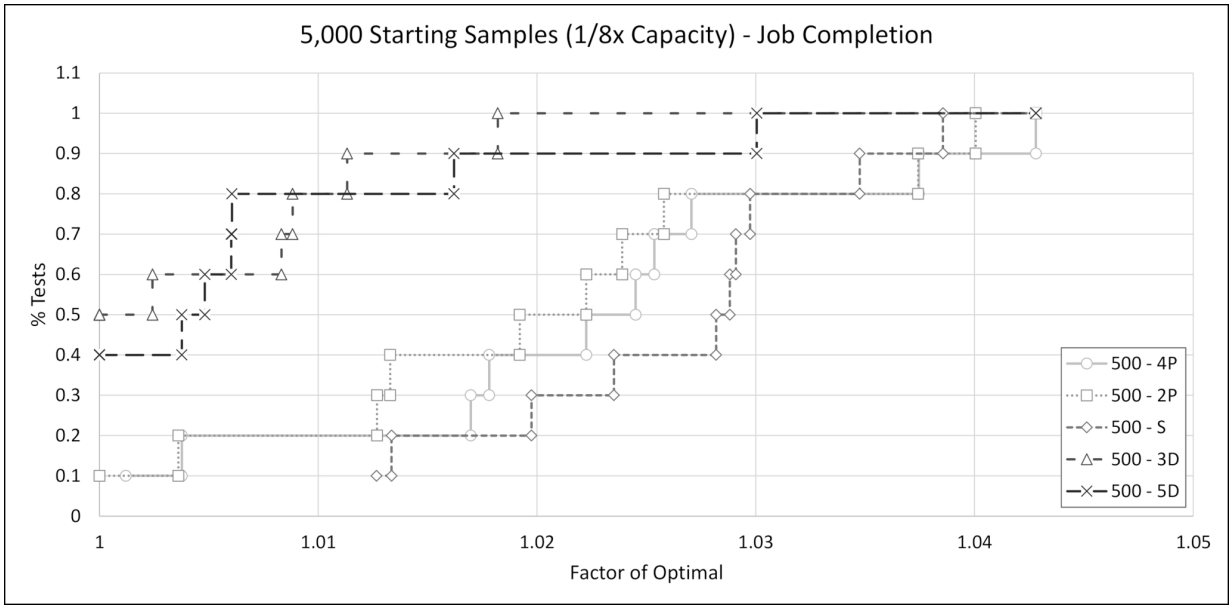


Figure 3.4: Performance profile comparing job completion among rescheduling policies for instances with one eighth the original capacity.

loads. When the capacity was between one half the original capacity and eight times the original capacity, the conclusions with respect to job throughput remained the same as when nominal parameter values were used as in section 3.5.1 and so we include these results in the Appendix B. When the plant capacity was lowered to one eighth of the original capacity, we observed that longer rescheduling policies outperformed frequent rescheduling, as shown in Figure 3.4. With greatly reduced plant capacity, there is more contention for resources and the longer horizons used by the "3D" and "5D" policies allows for better utilization of processes as discussed previously. This follows the same trend observed when increasing the load on the facility, although the results are more pronounced with the drastic reduction in capacity across all processes.

The makespan performance results remained largely the same as to those presented with moderate facility load in section 3.5.1. However, when the capacity was restricted to one quarter of the original capacity, the performance differences of the rescheduling policies decreases greatly. This is shown by the APF differences between the different policies compared to when the plant has full capacity, as shown in Table 3.6.

When measuring the proportion of jobs on time we observe that, for lead times of at least one week, and capacity one quarter or less than the original capacity, the policies with

Table 3.6: Average makespan APF differences comparison between one quarter capacity and full capacity instances.

ID	Average Makespan APF with Quarter Capacity	Average Makespan APF Differences with Quarter Capacity	Average Makespan APF with Full Capacity	Average Makespan APF Differences with Full Capacity
4P	1.0258	-	1.0	-
2P	1.0082	-0.0176	1.0705	0.0705
S	1.0309	0.0227	1.2422	0.1717
3D	1.0866	0.0557	1.66	0.4178
5D	1.1974	0.1108	2.0822	0.4222

longer horizons perform best. Figure 3.5 shows the proportion of jobs on time for each of the different policies with one eighth capacity and one month lead times. The results with one week lead times are similar to those with one month lead times, however in these cases the "5D" policy falls short of the other policies. This observation is likely caused by the increased latency between job arrival time and the first time a job may be scheduled, that the "5D" policy is subject to. The performance profiles for these other cases are included in the Appendix B. The results when capacity was increased beyond the original capacity did not differ substantially from those obtained using moderate facility load, presented in section 3.5.1.

These experiments demonstrate that, for plants which are starved for resources, the longer horizons used when scheduling less frequently can allow for better resource utilization. This better utilization can then be translated into better job completion performance and more jobs considered on time if long lead times are acceptable. These results differ from the general consensus in the literature that more frequent rescheduling is beneficial and show that plant and problem specific parameters, particularly with respect to facility load, may play an important role when choosing a suitable rescheduling policy.

3.5.4 Other Factors Considered

Beyond the results that were shown above, we also performed additional experiments simulating a plant that was run continuously for 24 hours each day as opposed to the 8 hours per day mode used for the experiments described above. For these tests we used the "4P", "2P", and "S" policies described in Table 3.1 but instead of using an 8 hour day

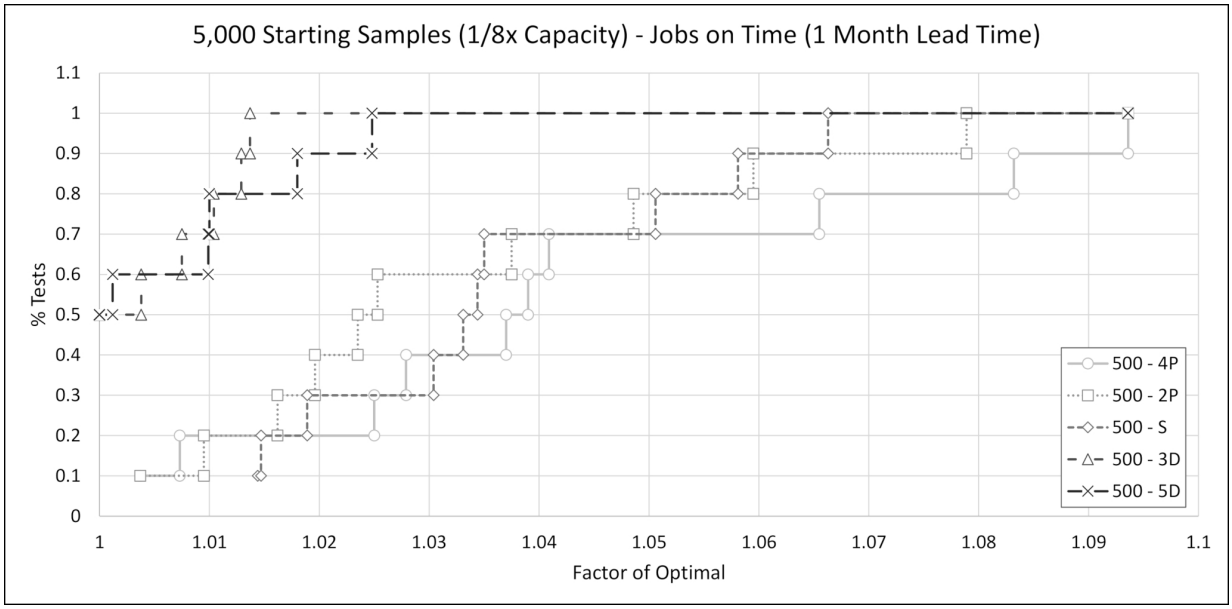


Figure 3.5: Proportion of jobs on time for one month lead times and instances with one eighth the original capacity.

with a 16 hour gap between horizons we used a 24 hour day with no gap. We also used a "2D" policy that is similar to the "3D", and "5D" policies which schedules two days at a time. This was done because solving three days and five days using a 24 hour horizon was computationally taxing and so we lessened the number of days down to two.

Additionally, when creating random jobs while generating test instances, the set of job paths that was sampled from was varied in some experiments. We tested sampling from paths that could be requested at the facility but did not necessarily arrive during the month of observation, and also paths that were completely random and may not be realistic to arrive at the facility.

In both of these cases, the conclusions drawn from these experiments did not change from those discussed when considering the experiments with nominal parameter values. The results were very similar for these experiments, with the most notable difference being that the performance differences between policies was less than when the plant was run with actual paths that may arrive at the facility. A possible explanation is that the increased diversity in paths spreads the process demand over more processes, and hence leads to less resource contention. Since the results were very similar and the conclusions do not change, we omit these results. The results of these experiments suggest that our previous results

did not rely on our choice of plant operating mode nor choice of job paths to sample from.

3.6 Chapter Summary

In this chapter, we presented the rolling horizon routine used to link individual horizons together for carrying out our long term experiments. The performance of using various rescheduling policies to schedule operations for a real industrial-scale facility were compared over a span of several months. The results demonstrate that choosing a suitable rescheduling policy can vary on the properties of the facility that is being scheduled. In particular, we found that the load on the facility and the processing capacity of the facility can have a large impact on selecting a rescheduling policy. When Under low to moderate loads, our results agreed with the most common conclusion in the literature: that rescheduling more frequently is better. However, when capacity is the main limiting factor, rescheduling less frequently may be beneficial.

Chapter 4

Study on Dynamic Timepoint Schemes

In this chapter we go into detail about the research that was conducted with respect to modifying the underlying time grids of our problem using iterative methods. Nearly all the research using non uniform discrete time grids have assumed that the time grids are fixed. However, as mentioned in section 2.5, choosing the time grids is non trivial and can lead to both wasted timepoints at times during the horizon where they are unnecessary, and missing timepoints whose inclusion would benefit the model. By shaping the time grids iteratively, we aim to improve solution quality with small computational overhead. We believe that the method developed in this chapter may be used for more general models that use a time layered graph representation. Therefore, we begin by presenting the more general model that is used in this chapter and explaining the differences between this model and ($P1$) (presented in section 2.3) in section 4.1. We then discuss the framework that we use to solve our problem in section 4.2. This includes the overall iterative method along with the heuristics that were used for our work to identify potentially beneficial missing timepoints to add, and possibly unneeded timepoints to remove. Section 4.3 describes the experiments that were carried out, how timepoint modification strategies were evaluated, and the results of the experiments. We go into detail discussing under what conditions these iterative methods appear to be most beneficial and under what conditions one may benefit more from using a fixed time grid.

4.1 Problem Description

In this section we discuss the scheduling problem that is under consideration for chapter 4. The problem is very similar to $(P1)$ presented in section 2.3, however we take a more network oriented approach to the problem and define it with more generality to emphasize the applicability of our proposed method to other applications that use time layered graphs.

Assume that we start with a time layered graph $G^* = (V^*, A^*)$ with underlying network $G = (V, A)$, timepoint sets ε, and travel function τ, as described in section 2.2. We have a set I of jobs that must be scheduled. Each job $i \in I$ is comprised of a discrete set of materials and a path $\Pi(i) = (\Pi(i,1), \Pi(i,2), \ldots, \Pi(i,m))$ of states, which it must be routed through, in order. We denote the number of materials of job i that arrive at process u at time $t \in \varepsilon(u)$ as $\alpha(i,u,t)$.

States of the underlying graph, $u \in V$, have a number of properties. We denote the number of resources of state u as $\rho(u)$, and the capacity of each resource as $\kappa(u)$. These properties restrict how material may flow through the network; for instance, sending m units of material from state u to a different state v at time t requires that $\lceil m/\kappa(u) \rceil$ resources of state u are available to be dispatched at time t. Note that we assume that packing material into state resources is not an issue as we allow material to be processed by different state resources and state resources to transport material from different jobs, as long as there is sufficient capacity. We assume that arcs that start and end at the same state, which represent material waiting at a state, have unlimited capacity and do not require any state resource allocation as the material is just being *held* at the state.

Resources of state u that are dispatched to state v at time t, become available to be reused after some time $\omega(u,v,t)$. We call $\omega(u,v,t)$ the return time of u to v at t. Let us define $\omega^{max}(u,t) = \max_{v \in N_G^+(u)} \omega(u,v,t)$, to be the longest return time for a u state resource which is utilized at time t. Similarly, let $\omega^{min}(u,t) = \min_{v \in N_G^+(u)} \omega(u,v,t)$ denote the minimum return time for a u state resource which is utilized at time t.

Let $H(i,u,t)$ denote the head of the arc in G^* corresponding to material of job i at state u being sent to the subsequent state in its path at time t. To make this notation more clear, let us again consider the example given in Section 2.2, with the corresponding figure repeated as figure 4.1. Suppose the path of job i is $\Pi(i) = (A, B, D)$ and the path of job j is $\Pi(i) = (A, C, D)$. Then we have $H(i, A, 1) = (B, 3)$, $H(j, A, 1) = (C, 2)$, $H(j, A, 2) = (C, 3)$, and $H(i, B, 1) = H(i, B, 2) = H(j, C, 2) = (D, 3)$.

We have three types of variables in the present problem. Let $y(u,v,t)$ denote the number of resources of state u dispatched to state v at time t. This allocation allows up to $\kappa(u)y(u,v,t)$ units of material to be sent from state u to state v at time t. Let $x(i,u,t)$

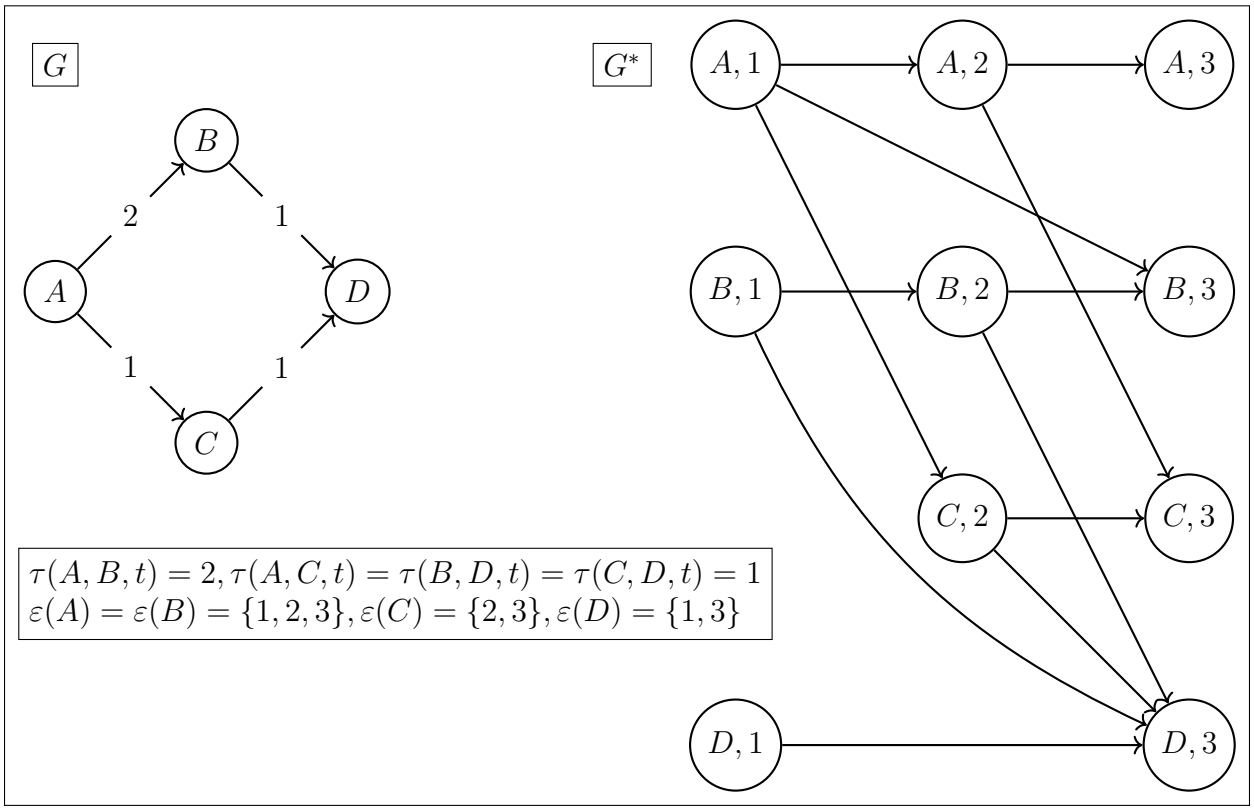


Figure 4.1: An example of a time layered graph.

denote the number of materials of job i sent from state u at time t to $H(i,u,t)$. Let $w(i,u,t)$ denote the number of materials of job i that wait at state u at time t, i.e. instead of proceeding to $H(i,u,t)$, the materials proceed to $(u, n(u,t+1))$. Note that again the x and w variables are flow variables on the arcs of G^*, and the resulting problem $(P2)$ (shown below) is a flow problem.

We now present the complete model formulation as problem $(P2)$.

$$\max_{x,y} \quad f_{G^*} = \sum_{i\in I}\sum_{u\in\Pi(i)}\sum_{t\in\varepsilon(u)} f(i,u,t)x(i,u,t) - \sum_{u\in V}\sum_{v\in N_G^+(u)}\sum_{t\in\varepsilon(u)} c(u,v,t)y(u,v,t) \quad (P2)$$

s.t.

$$x(i,u,t) + w(i,u,t) - \sum_{(w,t'):H(i,w,t')=(u,t)} x(i,w,t') = \alpha(i,u,t), \quad \forall\, i\in I, u\in\Pi(i), t\in\varepsilon(u) \quad (1)$$

$$\sum_{v\in N_G^+(u),t':t'\le t<t'+\omega(u,v,t')} y(u,v,t') \le \rho(u), \qquad \forall\,(u,t)\in V^* \quad (2)$$

$$\sum_{i\in I:\exists k,\Pi(i,k)=u,\Pi(i,k+1)=v} x(i,u,t) - \kappa(u)y(u,v,t) \le 0, \qquad \forall\,(u,t)\in V^*, v\in N_G^+(u) \quad (3)$$

$$\sum_{i\in I:\exists k,\Pi(i,k)=u,\Pi(i,k+1)=v} x(i,u,t) - \kappa(u)(y(u,v,t)-1) \ge 1, \qquad \forall\,(u,t)\in V^*, v\in N_G^+(u) \quad (4)$$

$$x(i,u,t) \ge 0, \qquad \forall\, i\in I, u\in\Pi(i), t\in\varepsilon(u) \quad (5)$$

$$w(i,u,t) \ge 0, \qquad \forall\, i\in I, u\in\Pi(i), t\in\varepsilon(u) \quad (6)$$

$$y(u,v,t) \ge 0, \qquad \forall\,(u,t)\in V^*, v\in N_G^+(u) \quad (7)$$

$$x, w, y \text{ integral}, \quad (8)$$

Constraints (1) are flow constraints which ensure that the in-flow that comes into a node (u,t) is equal to the out-flow that leaves (u,t) for each job i. Constraints (2) ensure that state resources are not over-allocated at any point in time. Constraints (3) ensure that enough state u resources are allocated at time t to support the amount of material scheduled to leave. Constraints (4) force that we do not dispatch resources without need, i.e. we do not waste state resources in our solution. Constraints (5) - (8) are non-negativity and integrality constraints for the variables. Note that both the w and y variables are completely determined by the values of the x variables; hence, for the remainder of the chapter, we will refer to the solution as x and but assume that the w and y variables are stored and accessible as well.

The objective function is a general weighted sum on the x and y variables, where $f(i,u,t)$ is the per material objective weight of sending material of job i from state u at time t to $H(i,u,t)$, and $c(u,v,t)$ is the per resource cost of dispatching resources of state u to state v at time t. We call a solution x', a "backward time-shifted" version of x, if x, and x' are solutions to $(P2)$ such that x' can be obtained from x by shifting some amount of material, δ, from a job i, scheduled to leave state u at time t to an earlier time t'. More precisely, $\exists i \in I, u \in \Pi(i), t, t' \in \varepsilon(u), \delta \in \mathbb{Z}, \delta > 0$ such that $t' \leq t$, $x'(i,u,t') = x(i,u,t') + \delta$, $x'(i,u,t) = x(i,u,t) - \delta$, and $x'(i,u,t) = x(i,u,t)$ otherwise. The only assumption that is considered on the objective function is that if x' is a backward time-shifted version of x, then $f_{G^*}(x') \geq f_{G^*}(x)$, i.e. the backward time-shifted solution is no worse than the original solution. We note that this is a fairly reasonable assumption as it is natural to assume that adding delays to a schedule will not improve its objective value.

This model may be adapted for use with similar scheduling problems to our application. For example, consider the problem of scheduling trains to travel between railroad stations, with the objective of transporting cargo from an origin station to a destination station. Let states be stations, and the resources of a state be trains at the station. The flow conservation constraints of our model ensure that cargo is routed through its specified path from the origin to the destination, while the resource allocation constraints ensure that we do not dispatch more trains than are available at each station.

Note that other types of problems may also be encoded using time layered graphs as well. For instance, buses are the state resources, states are bus terminals, and we want to impose that at least k buses are dispatched from terminal u to terminal v between times t_1 and t_2 ($\sum_{t_1 \leq t \leq t_2} y(u,v,t) \geq k$); materials are samples, states are machines that process samples, and we want to impose that if material is processed on machine u, then it is also processed subsequently on machine v within k minutes after the completion of machine u ($x(i,u,t) \leq \sum_{t+\tau(u,v,t) \leq t' \leq t+\tau(u,v,t)+k} x(i,v,t') \ \forall i \in I, t \in \varepsilon(u)$, and $w(i,v,t) + x(i,v,t) \leq \sum_{t-\tau(u,v,t')-k \leq t' \leq t-\tau(u,v,t')} x(i,u,t') \ \forall i \in I, t \in \varepsilon(v)$).

The main differences between problem $(P2)$ and problem $(P1)$, presented in section 2.3, are that it is no longer assumed that states (previously processes) have the same processing time to reach subsequent states, and that we do not assume that state (process) resources become available immediately after they finish processing. To demonstrate why relaxing these assumptions may be useful, consider again the problem of scheduling trains. Without the first relaxation, we would need that a train leaving a station takes the same amount of time to reach any other station, which is likely not the case. The second relaxation allows us to have a train arrive back at the origin station after some time which depends on both the station it was sent to, and the time at which it left the station, i.e. without

this relaxation we would have that trains may depart the origin station again as soon as they reach their destination.

To demonstrate that $(P2)$ is a more general version of $(P1)$, we give an overview of how to transform an instance of $(P1)$ into an instance of $(P2)$. Firstly, set the following parameters for $(P2)$ to be equal to those for $(P1)$: I, V, ε, κ, ρ, Π, α, f, c. Construct the arc set A of $(P2)$ as follows: for each job i, for each pair of adjacent processes (u, v) in i's path, add (u, v) to A. For each state u, set $\tau(u, v, t)$ for $(P2)$ to $\tau(u)$ for each neighbor v and timepoint t. For each state u, set $\omega(u, v, t)$ to the processing time $\tau(u)$ for each neighbor v and timepoint t. Now, any solution which satisfies the problem $(P2)$, satisfies the same operational constraints as those imposed on $(P1)$, namely: flow constraints, resource allocation constraints, and resource capacity constraints. In particular, using the above transformation, optimal schedules for both problems $(P1)$ and $(P2)$ will be equivalent.

4.2 Timepoint Modification Framework

In this section we discuss the framework for solving our scheduling problem $(P2)$ from section 4.1, using an iterative approach. We first discuss the work of Boland et al. [3] and briefly explain why the algorithm and results presented there may not be directly applied to our scheduling problem. They concern themselves with a similar scheduling problem of transporting commodities through a network by dispatching trucks or trailers between depots, with the goal of transporting all commodities from their origin depot to its destination depot before it is due. They define a relaxation to the continuous time version of their problem (assuming time is discretized into one minute intervals), and show that at each iteration after solving their relaxation, either they are able to convert the solution into one which is feasible for the continuous time version of their problem, or they are able to refine their relaxation by adding a new timepoint and modifying their time layered graph. Since there are finitely many timepoints to add, eventually their relaxation will model the continuous time version and solving this iteration will be sufficient to get an optimal solution to the continuous time version. Attempting to apply the construction of their relaxation problem on our application does not result in the same relaxation. We attempted to modify their procedure to prove a result similar to theirs but were unsuccessful. However, our procedure can still produce beneficial results even if we cannot prove that it terminates with the optimal solution to the continuous time formulation. In their problem, they have trucks or trailers which travel between depots. In our case, these trailers correspond to state resources, and the depots correspond to states. The main difference between our

problems comes from the assumption that in their case, the number of trucks or trailers that can be dispatched from a depot is unbounded ($\rho(u) = \infty$), which allows them to transform a solution to their continuous time version of their problem into a solution for their relaxation of equal cost. Essentially, this corresponds to us removing constraints (2) from model ($P2$). However, these constraints present a roadblock in using their methods on our application. In lieu of this, the general idea of identifying timepoints that need to be added and repeating this procedure iteratively is used below and has similarities to the approach of Boland et al.

We first present Figure 4.2 to give an overview of how the framework proceeds, before describing the framework in more detail below. Note that some details about the framework are omitted in the flowchart to avoid over complicating the diagram. Namely, we omit the details that problems are given solutions found from previous iterations as input and how the previously found solutions are transformed into feasible solutions for the current problems. Note that this transformation is simple and roughly involves copying the x variables from the original solution to the x variables for the new solution, and then setting the w and y variables for the new solution appropriately so that the new solution is feasible.

Our iterative method is now described as follows. We begin by instantiating our timepoint sets, ε, to some sufficiently coarse uniform discretization which we will call `start_disc`. We then use a solution method, e.g. passing ($P2$) to Gurobi, or CPLEX, to solve problem ($P2$) and obtain a list, X, of any feasible solutions found by the optimization method. Using each obtained solution $x \in X$ as input, we proposed a set of algorithms, i.e. Get Instant Start Timepoints, Get Overloaded Timepoints, and Get Dominated Timepoints (which will be discussed in more detail later) to obtain a set of timepoints ε^+ to add to our current timepoints and a set of timepoints ε^- to remove from our current timepoints. That is, algorithms Get Instant Start Timepoints and Get Overloaded Timepoints produce a set of timepoints to add for each solution $x \in X$, and Get Dominated Timepoints produces a set of timepoints to remove for each $x \in X$. To obtain ε^+, we take the union of all of the sets of timepoints to add, and to obtain ε^-, we take the intersection of all of the sets of timepoints to remove. By choosing to use the union for added timepoints, we add all timepoints that are identified as being potentially beneficial, and by choosing to use the intersection for removed timepoints, we remove only those timepoints which were considered not needed for every solution $x \in X$. We proceed to construct a new instance of problem ($P2$), ($P2^*$), by adding ε^+ and/or removing ε^- from the current timepoints ε. The best solution found previously, x, is used to generate a new solution x^* which is feasible for our newly formed problem, ($P2^*$). x^* is given as an initial solution to our new problem ($P2^*$) and we repeat this process of solving the current problem, using the

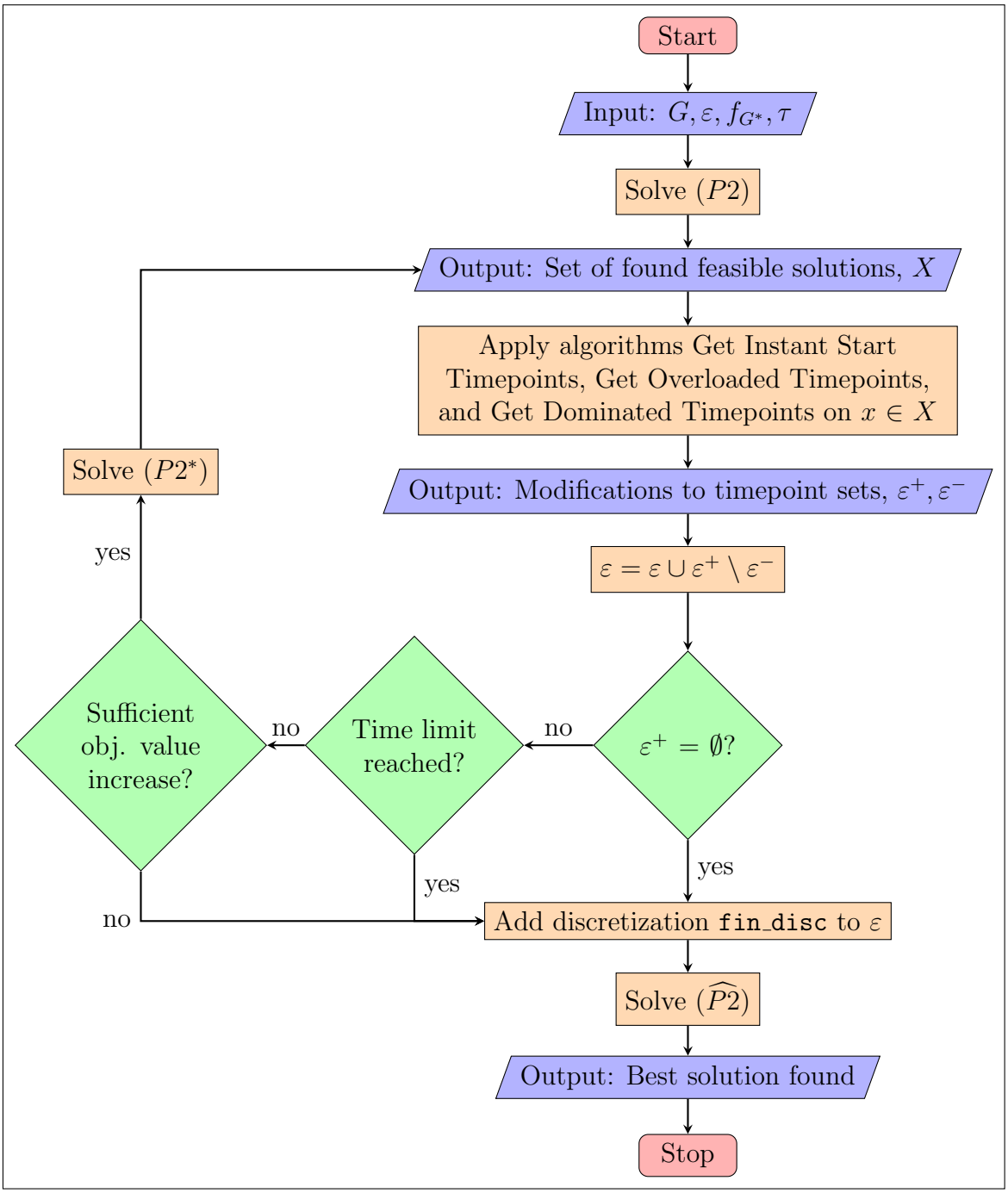


Figure 4.2: A flowchart outlining the dynamic timepoint framework.

solution(s) found to generate new timepoints to add and remove, and then modifying the current set of timepoints. We provide the solution x^* as an initial solution for $(P2^*)$ so that the solver may use x^* as a feasible solution, with the goal of optimizing $(P2^*)$ more quickly than if no initial solution was provided. This continues until we reach a stopping criterion such as reaching a computational time limit, finishing an iteration such that algorithms Get Instant Start Timepoints and Get Overloaded Timepoints do not produce any new timepoints to add, or completing an iteration in which there is insufficient improvement between the new solution and the previous solution. Note that we measure the difference between solution values as the new solution value divided by the old solution value and we call the improvement we require between iterations `obj_thresh`. To demonstrate, if `obj_thresh` = 1.1, then we impose that the new solution value is at least 1.1 times the old solution value, and if an iteration terminates such that this condition is not met, we stop iterating. We call the allowed time between solutions `sols_tl`, and the time limit for the aforementioned iterative procedure `its_tl`. That is, if we are solving a single instance of $(P2)$ using a method that generates multiple solutions such as CPLEX or Gurobi, then once we find an incumbent solution to $(P2)$ we require that a better solution is found before `sols_tl` seconds have elapsed since finding the previous solution. If no better solution is found during this time, we quit the solution procedure for the iteration and return with whatever solution(s) were found. This is done with the aim of reducing the amount of time spent on a single iteration. The time limit for the entire iterative procedure, `its_tl`, is checked between iterations and if exceeded, we stop iterating and move on to the next step. The way this was implemented in our experiments was to call Gurobi with the remaining time left of the iterative procedure (`its_tl` - (current time - start time)) as the time limit and to provide two types of callbacks to Gurobi. The first type of callbacks are triggered any time a new solution is found; it records the time that the latest solution was found and stores the solution in an external array for use with the timepoint modifying algorithms later. The second type of callbacks are triggered intermittently and check how long it has been since the last solution was found. If the elapsed time has exceeded `sols_tl`, then we quit Gurobi and proceed to the next iteration.

After this process of adding and removing timepoints iteratively has reached a stopping criteria and terminated, we store the best solution found previously, $\hat{x}$, add all of the timepoints associated with some chosen discretization (which we call `fin_disc`) to our timepoint sets, create a problem $(\widehat{P2})$, and solve this problem with input $\hat{x}$ given as an initial solution. We call the time limit passed to the solution method for solving this final problem `fin_tl`. By adding all of the timepoints from a discretization which is assumed to provide acceptable solutions, we aim to find any solutions better than $\hat{x}$ we may have missed earlier during our iterative process. However, despite adding possibly many new timepoints

Table 4.1: Description of parameters used for the framework.

Parameter Name	Parameter Description
`start_disc`	The timepoint discretization to use for building the first iteration of problem $(P2)$
`fin_disc`	The timepoint discretization to use for building the final solve of problem $(P2)$, after the iterative procedure
`its_tl`	The total amount of time to allow for the iterative procedure (before adding `fin_disc` to $(P2)$ and solving $(\widehat{P2})$)
`obj_thresh`	The required objective improvement between iterations, measured as new objective value as a factor of old objective value
`sols_tl`	The allowed time between solutions during a single iteration. If more than `sols_tl` time has passed since finding the last solution, then quit the current iteration
`fin_tl`	The allowed time for solving the final problem $(\widehat{P2})$, after adding `fin_disc` to $(P2)$

during this final step, we still expect to take advantage of the incumbent solution $\hat{x}$, to be able to optimize $(\widehat{P2})$ faster than if we optimized $(\widehat{P2})$ from scratch. The framework terminates by returning the best solution found for problem $(\widehat{P2})$.

Table 4.1 provides a summary of the parameters used in the framework with their corresponding descriptions for reference.

We include and discuss the heuristics used to identify new timepoints to add and which timepoints to remove below. However, for more information about the framework, we provide pseudo code algorithms for the framework and how to update solutions between iterations in the Appendix C.

Algorithms Get Instant Start Timepoints, Get Overloaded Timepoints, and Get Dominated Timepoints describe the heuristics that are used for modifying timepoints during the procedure. The driving idea of the heuristics is to add timepoints such that actions from a previous schedule may be shifted to happen earlier in time, and to remedy cases where a state's resources cannot be efficiently utilized because of a lack of timepoint availability. Similarly, we remove timepoints which seem to be unnecessary, from a time-shifting perspective, to reduce the size of the resulting problems. Our goal is that by applying these heuristics iteratively, we may shape the starting coarse timepoint sets into sets which include sufficient timepoints for obtaining high quality solutions, but without the unneeded

timepoints that a fine uniform discretization may have. Note that this is where we use the assumption that the objective function of $(P2)$ is such that backward time-shifting solutions does not result in worse objective values. If this assumption holds, then we expect that shifting scheduled events to happen earlier in time (possibly allowing for more events to be scheduled) will improve the best objective value over each iteration.

Algorithm Get Instant Start Timepoints shown in Table 4.2 describes how timepoints are added such that material that leaves from state a w to a state u may leave u upon arrival. Namely, suppose a state u has resources dispatched at timepoint t ($\sum_{v \in N_G^+(u)} y(u,v,t) > 0$). We then consider all vertices (w,t') of G^* that have an arc from (w,t') to (u,t) that is being used ($y(w,u,t') > 0$) and whose resources will arrive before time t ($t' + \tau(w,u,t') < t$). For each of these neighbors, we add a new timepoint at the actual time that samples departing from this vertex arrive at state u (i.e. $t' + \tau(w,ut')$ is added to $\varepsilon(u)$). This procedure is carried out over all vertices (u,t) of G^* such that resources of state u are dispatched at time t.

We call the timepoints added by this heuristic, "instant start timepoints" as they allow material which leaves a state w and arrives at a state u to begin departing u immediately upon arrival ($t' + \tau(w,u,t')$) instead of waiting until time t to leave. The timepoints that this heuristic adds allows material which is being scheduled by the model to have fewer instances where it must wait at a state before proceeding to the next one in its path. We anticipate that by including these timepoints, future schedules may obtain greater objective value by shifting operations to happen earlier, possibly allowing more operations to be scheduled later in the horizon. Figure 4.3 provides a graphical representation of how instant start timepoints are identified and added to the timepoint sets.

Algorithm Get Overloaded Timepoints shown in Table 4.3 describes how timepoints are added for states which are heavily utilized but whose timepoint sets is lacking potentially useful timepoints. Suppose a state u has all of its resources dispatched at time t ($\sum_{v \in N_G^+(u)} y(u,v,t) = \rho(u)$). We use this criteria to identify times of high resource utilization for state u. We then add a timepoint for state u at the earliest time that we can guarantee that the resources will be available again, that is after the maximum return time of u at time t, $t + \omega^{max}(u,t)$. We continue to repeat adding timepoints spaced by $\omega^{max}(u,t)$ until we reach the timepoint proceeding t, $n(u,t+1)$. Note that this heuristic will only affect timepoints where there is extra time between when a resource for state u will return to be used again, and the next timepoint dispatching may occur at ($t + \omega^{max}(u,t) < n(u,t+1)$). We carry out this procedure over all vertices (u,t) of G^* such that all resources of state u are dispatched at time t.

We call the timepoints added by this heuristic "overloaded timepoints" because they

Table 4.2: Algorithm 2, Get Instant Start Timepoints

Algorithm 2 Get Instant Start Timepoints

function GET INSTANT START TIMEPOINTS(G, G^*, ε, x)
  $\varepsilon^+ \leftarrow \{\varepsilon^+(u) = \emptyset : u \in V\}$ ▷ Instantiate set of timepoints to add
  for all $u \in V$ **do**
    for all $t \in \varepsilon(u)$ **do**
      if $\sum_{v \in N_G^+(u)} y(u, v, t) > 0$ **then** ▷ Material is leaving (u, t) in the solution
        for all $(w, t') \in N_{G^*}^-((u, t)) : t' + \tau(w, u, t') < t$ **do** ▷ Consider neighbors such that material departing from that neighbor arrives at u before t
          if $y(w, u, t') > 0$ **then** ▷ Arc from (w, t') to (u, t) that is being used in solution
            $\varepsilon^+(u) \leftarrow \varepsilon^+(u) \cup \{t' + \tau(w, u, t')\}$ ▷ Set $t' + \tau(w, u, t')$ to be added to $\varepsilon(u)$
          end if
        end for
      end if
    end for
    $\varepsilon^+ \leftarrow \varepsilon^+ \cup \{\varepsilon^+(u)\}$
  end for
  return ε^+ ▷ Return set of instant start timepoints to add
end function

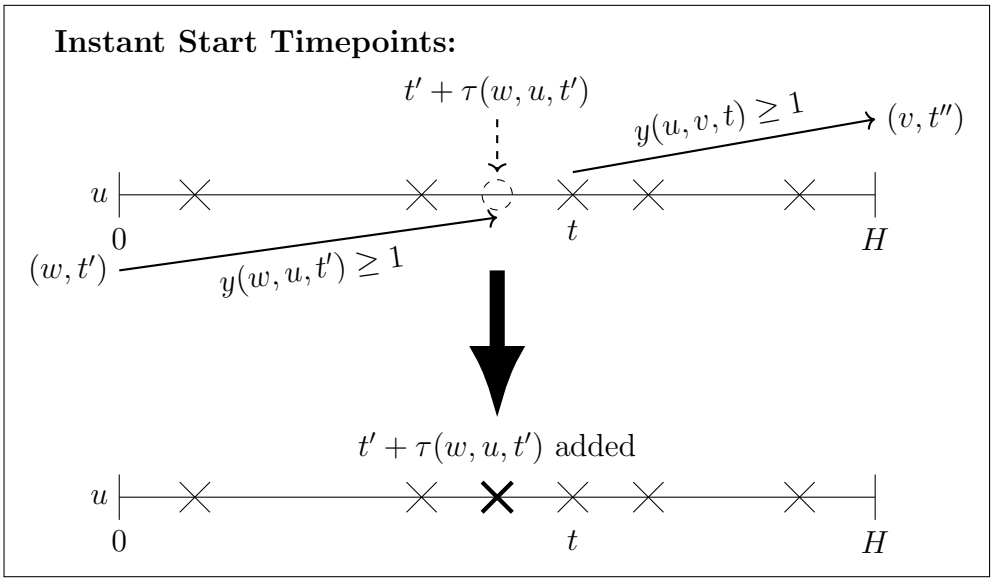


Figure 4.3: A figure demonstrating under what conditions we add new instant start timepoints.

are added in cases where we identify a fully utilized resource. This heuristic aims to reduce cases where there is a backlog of materials waiting at a state, but the state's resources are underutilized because u has insufficient timepoints around time t. In these cases, we add timepoints for state u so that resources for state u may be used once they return from a neighboring state and become available again, thereby allowing better resource utilization for this state. Figure 4.4 demonstrates how overloaded timepoints are added to the timepoint sets.

Algorithm Get Dominated Timepoints shown in Table 4.4 describes how timepoints are removed in cases when we have two adjacent timepoints that are sufficiently close. Consider a state u has two adjacent timepoints, t and $n(u, t+1)$, such that no resources of state u are dispatched at time $n(u, t+1)$ ($\sum_{v \in N_G^+(u)} y(u, v, n(u, t+1)) = 0$). Suppose that the timepoints are also close enough that a resource dispatched at time t may not be dispatched at time $n(u, t+1)$, that is the time difference between the points $(n(u, t+1) - t)$ is less than the minimum return time of u at time t, $\omega^{min}(u, t)$. Finally, if there are also no flows on arcs whose head is $(u, n(u, t+1))$ ($\sum_{(w,t') \in N_{G^*}^-((u,n(u,t+1)))} y(w, u, t') = 0$), and material that leaves state u at time t arrives at its destination before material which leaves u at time $n(u, t+1)$ ($t + \tau(u, v, t) \leq n(u, t+1) + \tau(u, v, n(u, t+1))$ $\forall v \in N_G^+(u)$), then we remove $n(u, t+1)$ from state u's timepoints.

Table 4.3: Algorithm 3, Get Overloaded Timepoints

Algorithm 3 Get Overloaded Timepoints

function GET OVERLOADED TIMEPOINTS(G, G^*, ε, x)
    $\varepsilon^+ \leftarrow \{\varepsilon^+(u) = \emptyset : u \in V\}$ ▷ Instantiate set of timepoints to add
    for all $u \in V$ **do**
        for all $t \in \varepsilon(u)$ **do**
            if $\sum_{v \in N_G^+(u)} y(u, v, t) = \rho(u)$ **then** ▷ Solution is dispatching all resources of state u at time t
                $t' \leftarrow n(u, t+1)$
                while $t + \omega^{max}(u, t) < t'$ **do** ▷ Current time is between t and proceeding timepoint
                    $t \leftarrow t + \omega^{max}(u, t)$
                    $\varepsilon^+(u) \leftarrow \varepsilon^+(u) \cup \{t\}$ ▷ Set t to be added to $\varepsilon(u)$
                end while
            end if
        end for
    end for
    return ε^+ ▷ Return set of overloaded timepoints to add
end function

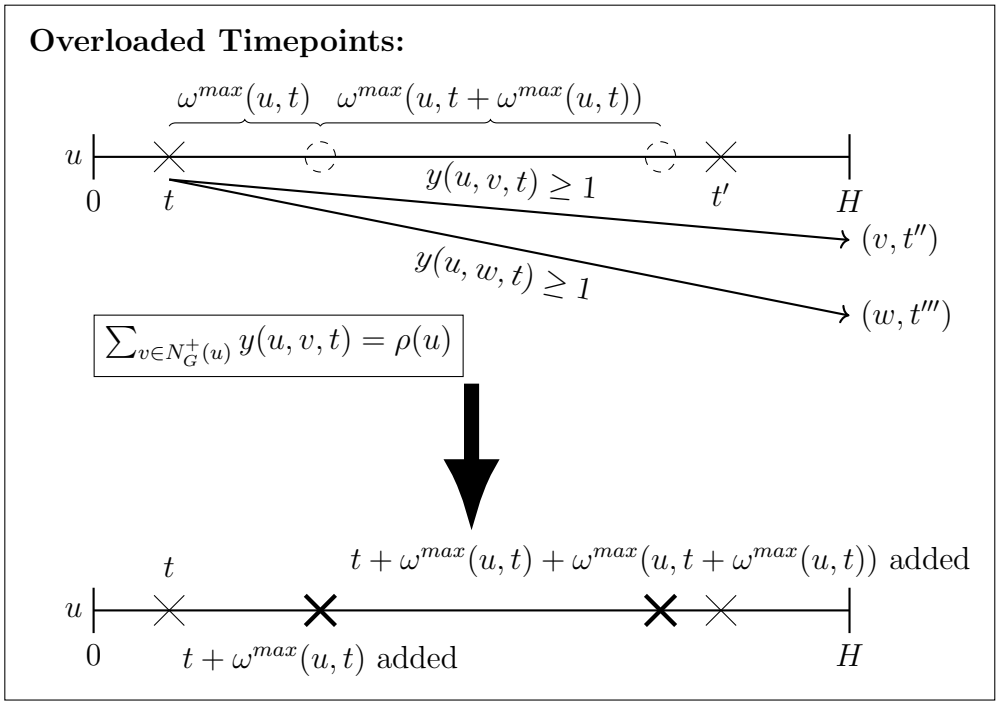


Figure 4.4: A figure demonstrating under what conditions we add new overloaded timepoints.

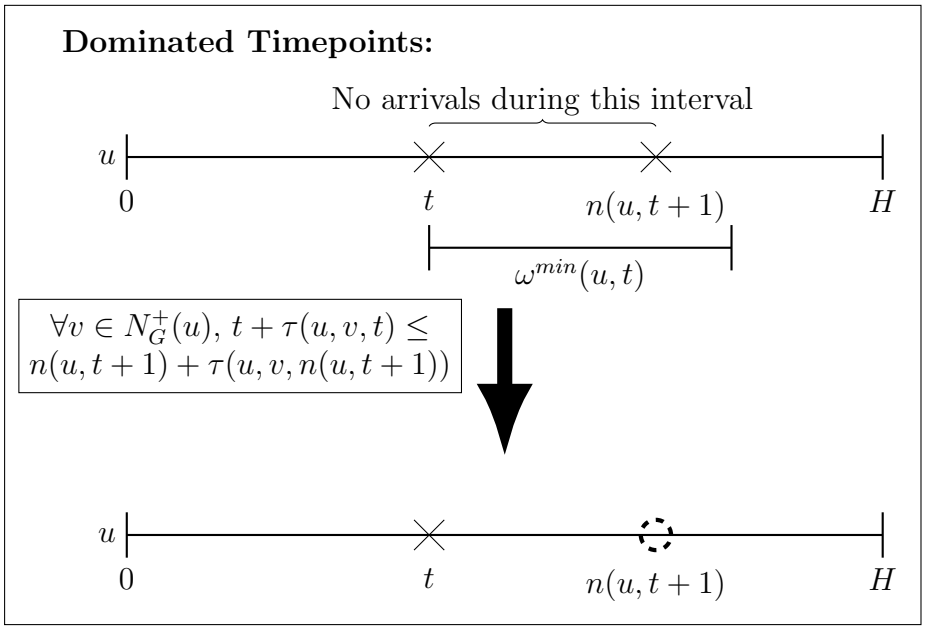


Figure 4.5: A figure demonstrating under what conditions we mark timepoints as dominated, for removal.

We call the timepoints which are removed according to the previous critera "dominated timepoints". Based on the assumption that backward time shifting a solution cannot worsen its objective value, then any solution which uses resources of state u at time $n(u, t+1)$ could be changed to an equal or better solution which uses the same resources at time t. Identifying and removing these timepoints helps to reduce the number of timepoints in the model, and hence the model size with the aim of improving the CPU cost of solving the model. Figure 4.5 shows how we identify dominated timepoints to remove from the timepoint sets.

Note that the proposed framework above also has several limitations. Given that this is a heuristic approach, global optimality is not guaranteed; thus, we accept practical solutions that may be sub-optimal but can be resolved in short CPU times. The algorithms Get Instant Start Timepoints, Get Overloaded Timepoints, and Get Dominated Timepoints used for modifying the timepoints between iterations have no guarantees, even though the ideas behind their development seems suitable and intuitive. With respect to the convergence of the framework, we did not observe any cycling (timepoints being added and subsequently removed many times) in our testing, but we do not guarantee that cycling may not occur. However, if such cycling is possible, the various stopping criteria

Table 4.4: Algorithm 4, Get Dominated Timepoints

Algorithm 4 Get Dominated Timepoints

function GET DOMINATED TIMEPOINTS(G, G^*, ε, x)
  $\varepsilon^- \leftarrow \{\varepsilon^-(u) = \emptyset : u \in V\}$ ▷ Instantiate set of timepoints to remove
  for all $u \in V$ **do**
    for all $t \in \varepsilon(u)$ **do**
      if $\sum_{v \in N_G^+(u)} y(u, v, n(u, t+1)) = 0$ **then** ▷ No materials leaving state u at time $n(u, t+1)$
        if $n(u, t+1) - t < \omega^{min}(u, t)$ **then** ▷ We cannot allocate the same resource to both timepoints
          if $t + \tau(u, v, t) \leq n(u, t+1) + \tau(u, v, n(u, t+1))\ \forall v \in N_G^+(u)$ **then** ▷ Material leaving at time t arrives before material leaving at time $n(u, t+1)$
            if $\sum_{(w,t') \in N_{G^*}^-((u,n(u,t+1)))} y(w, u, t') = 0$ **then** ▷ No material arrives after t and at or before $n(u, t+1)$
              $\varepsilon^-(u) \leftarrow \varepsilon^-(u) \cup \{n(u, t+1)\}$ ▷ Set $n(u, t+1)$ to be removed from $\varepsilon(u)$
            end if
          end if
        end if
      end if
    end for
  end for
  return ε^- ▷ Return set of dominated timepoints to remove
end function

discussed above will still ensure that the framework terminates.

Additionally, there are a number of parameters whose values must be chosen a priori such as the starting discretization to use for the initial problem (`start_disc`), the values for the stopping criteria (`its_tl`, `obj_thresh`, and `sols_tl`), and the parameters relating to solving ($\widehat{P2}$) (`fin_disc` and `fin_tl`). The choices for these parameters will have an influence on the performance of the framework, and setting these parameters will depend on the specific problem being solved and one's solving preferences (CPU time limitations, hardware limitations, required solution quality, etc.). We present the actual choices for the parameters used in our experiments in section 4.3.1.

4.3 Computational Experiments

We now discuss the experiments that were conducted using our previously described iterative method. In section 4.3.1 we discuss the choices of parameters for the iterative method used, and the other discretizations that will be used for comparison. Section 4.3.2 describes the instances that we use for our experiments, along with how each method was evaluated. Sections 4.3.3 - 4.3.6 include the results of the experiments, as well as discussion on when our iterative method provides the most benefit and when it may be better to forgo the iterative procedure.

4.3.1 Policies Tested

The framework described previously in section 4.2 does not specify parameters for stopping criteria to use for ending the iterative phase of the procedure. It allows setting a time limit for both the iterative phase and the final solve, an objective value percent improvement threshold between iterations, and a maximum time threshold between solutions when solving a model. Furthermore, it does not specify what discretization to use when constructing the problem to solve for the first iteration of the framework, or the discretization to use when the iterative process has finished and we add missing timepoints from a known "good" discretization. Setting these parameters to different values may change what timepoints are added to the timepoint sets, which may impact how the framework performs. Therefore, we must select some values for these parameters for testing, and the choice of these parameters is important when using our iterative method.

For the computational experiments described in this chapter, we used the following choices of parameters for testing the efficacy of our method. Similar to how we defined

Table 4.5: Descriptions of iterative policies tested.

Policy Name	Starting Discretization	Time Limit for Iterative Phase (s) (`its_tl`)	Min. Objective Change (`obj_thresh`)	Max. Time Between Solutions (s) (`sols_tl`)	Time Limit for Final Solve (s) (`fin_tl`)
5 - 0 - UD60	UD60	600	1	5	600
5 - 0 - UD120	UD120	600	1	5	600
5 - 0 - UD240	UD240	600	1	5	600
60 - 1.05 - UD240	UD240	600	1.05	60	600

the NUDM discretization, we will refer to the uniform discrete M (UDM) discretization as having a timepoint at times $S, S+M, S+2M, \ldots, S+\lfloor H/M \rfloor M$ for each resource, where S is the start time of the horizon and H is the length of the horizon. We test each of UD60, UD120, and UD240 as starting discretizations (`start_disc`) with no minimum percent improvement between iterations (`obj_thresh` = 1) and a five second time limit between solutions during a single iteration (`sols_tl` = 5 seconds). These choices represent a policy which favors doing many iterations without spending too much time on each one. We also test our method by starting with the UD240 discretization, allowing sixty seconds between solutions (`sols_tl` = 60 seconds), and requiring a minimum objective value improvement of 5% between iterations (`obj_thresh` = 1.05). This set of parameters represents a policy which is willing to spend more time searching for solutions during a solve, but also may not perform as many iterations if the amount of improvement slows. These parameter values were chosen experimentally, as the resulting policies performed best in our testing. All iterative policies were given a 10 minute time limit for both the iterative phase (`its_tl` = 10 minutes) and the final solve after adding the known good discretization (`fin_tl` = 10 minutes). The NUD60 discretization was chosen as the set of timepoints to add after the iterative process ends (`fin_disc` = NUD60) as it was shown to provide high quality solutions in prior experiments [22]. A summary of these policy parameters are shown in table 4.5.

These iterative policies were compared against using the following discretizations without any sort of modifications: UD60, UD120, UD240, NUD60. We will refer to the use of these discretizations without modification as *static discretizations* throughout this section. These discretizations were chosen because they provide a wide range of granularity in the timepoint sets and so they should allow us to observe the tradeoff between computational expense and solution quality between the different discretizations. Additionally, the

NUD60 discretization was shown to perform best among these other static discretizations in [22], and so we use it as a benchmark for comparing the iterative policies' performance.

4.3.2 Testing Procedure

Up to this point, we have described the problem as a scheduling problem over a time layered graph. We used this more general setting to emphasize that this method is not solely applicable to our use case and may be used for other applications such as railroad [23, 51] or truck routing [3], or job shop problems [4]. Our experiments however deal again with the case of the industrial sized, analytical services facility used in chapter 3. In particular, we have the following differences compared to the general description of the framework above. We have a single return time for u state resources used, independent of what time the resources are utilized and to which neighboring state they are dispatched to, $\omega(u, v, t) = \omega(u)\ \forall v \in N^+(u), t \in \varepsilon(u)$. We have a single travel time from state u to any other neighboring state v, again independent of what neighbor the material travels to and at what time, $\tau(u, v, t) = \tau(u)\ \forall v \in N^+(u), t \in \varepsilon(u)$. Furthermore, the state resources become available immediately after material has been moved from state u to state v, i.e. $\tau(u) = \omega(u)\ \forall u \in V$. The parameters of the facility such as processing times, number of process resources, processing capacity are the same as those described in section 3.4.

Regarding the objective function of $(P2)$, f_{G^*}, we use $f(i, u, t) = 1\ \forall i \in I, u \in \Pi(i), t \in \varepsilon(u)$, and $c(u, v, t) = 0\ \forall u \in V, v \in N^+(u), t \in \varepsilon(u)$. These choices for f and c were used so that we may maximize the total amount of material that is being sent through the network assuming that the resource usage costs are negligible. To evaluate the efficacy of the framework described previously in section 4.2 we use the following procedure. For each policy that is being tested, we record the maximum objective value that the policy has obtained after 1 minute, 5 minutes, 15 minutes, 30 minutes, and 60 minutes have elapsed. We will refer to these elapsed times as "checkpoints". To normalize the results between instances, for each policy and checkpoint, we compare the objective value at that checkpoint as a percentage of the maximum objective value any policy obtains over the entire procedure. This allows us to better visualize the gain in objective value obtained by allocating larger amounts of computational time to the solver and also allows us to more easily compare the performance of the different policies at each checkpoint. The times indicated above were chosen so that we may try to compare based on a wide range of solving requirements and based on the observed solving times of our problems. Additionally, we also record the amount of time taken for each policy to complete the procedure and report this value as a proportion of the maximum time taken over all policies. An example of how to interpret the results is given in section 4.3.3.

Table 4.6: Sizes of instances in each category.

Size Category	Min. Variables	Min. Constraints	Max. Variables	Max. Constraints	Instances
Small	304,444	209,449	698,866	388,218	10
Medium	1,146,524	614,159	1,997,392	1,108,319	34
Large	2,016,628	1,117,913	3,399,906	1,880,512	16

We sort our results into three categories based on size. We measure instance size in terms of number of variables and constraints present in the model created when using the NUD60 static discretization. Instances were categorized into small, medium, and large size problems. Instances with less than 1,000,000 variables were considered to be small, instances with at least 2,000,000 variables were considered to be large, and other instances were considered medium sized. Note that all of the variables in our model are integer variables, and hence these numbers (1,000,000 and 2,000,000) refer to the total number of (integer) variables in the instance. More information about the sizes of the instances in each category is shown in table 4.6. Columns 2 and 3 present the minimum number of variables and constraints among all instances in each category. Similarly, columns 4 and 5 present the maximum number of variables and constraints among all instances. Column 6 indicates how many instances were sorted into each size category. Note that the number of instances sorted into each category varies from 10 to 34. The reason for this is because the instances were first formulated based on realistic horizon lengths and job arrivals using data from the indutrial partner. The instances were not sorted based on size until the results were analyzed, at which point this discrepancy was observed.

The individual instances are made up of various combinations of horizon length, and number of samples. The number of days varies from one to seven. Regarding the actual plant under analysis, the number of samples considered varies between 5,000 and 30,000. Each instance was solved assuming that the facility was able to operate constantly over the scheduling horizon to simplify testing. Detailed information about each instance including model size, relative objective values at each checkpoint, and which size category each instance was sorted into is provided in Appendix D.

Tests were run using 2 threads running at 3.2 GHz on a machine with access to 8 GB of RAM. The implementation was done using the Julia programming language (version 0.6.2), along with the Gurobi.jl (version 0.4.1) and JuMP.jl [10] (0.18.2) packages. Gurobi (version 8.0.1) [16] was used for solving the problems. The optimality tolerance given to Gurobi was set to stop solving when the best found solution was within 0.5% of the optimal solution.

4.3.3 Performance on Small Size Problems

The experiments described in this section consist of instances which were categorized as being small. Figure 4.6 depicts the results over the small instances. By observing the values for each policy after 1 minute has elapsed, we conclude that after 1 minute, we could obtain a solution whose value is approximately 54% of the highest objective values among all tested policies by using a UD240 static discretization. Similarly, static UD120, UD60, and NUD60 discretizations yield approximately 77%, 89%, and 98% respectively of the greatest objective value after 1 minute. These values demonstrate that as the granularity of the static policies is increased, so does the performance of the schedule obtained from these policies.

By examining the results for the dynamic timepoint policies, we observe that the performance obtained using any of the dynamic variants is approximately equal to that of the NUD60 after 1 minute has elapsed. Furthermore, the performance of the dynamic timepoint policies increases, surpassing the NUD60 policy, after 5 minutes have elapsed. However, the final cluster of bars depicting the amount of time required for each policy to finish demonstrates that on these instances, the dynamic policies take much longer to terminate compared to the static policies. This extra time may be attributed to the iterative process of adding timepoints and solving several models. From a tradeoff standpoint, the static NUD60 discretization offers nearly the best performance, at a fraction of the computational cost that the dynamic policies have.

This efficient tradeoff between objective value and solving time was presented in [22] and demonstrates that when instances are small, the NUD60 discretization is a good performer on the present scheduling problem. In these cases where the problem size is relatively small and the NUD60 policy may finish quickly, the dynamic framework appears to be unnecessary as it may improve the solution quality only marginally but may also incur a comparatively large computational cost.

4.3.4 Performance on Medium Size Problems

The results on the set of medium sized instances presented by Figure 4.7 indicate a different story, however. Observing the performance after 1 minute, we no longer see the NUD60 discretization being the best performer. In fact, the UD240 and UD120 static discretizations both have performance better than the UD60 and NUD60 static policies after 1 minute. This can be attributed to the instances being larger, and hence the solver not being able to make as much progress as quickly, on the problems with more dense timepoint

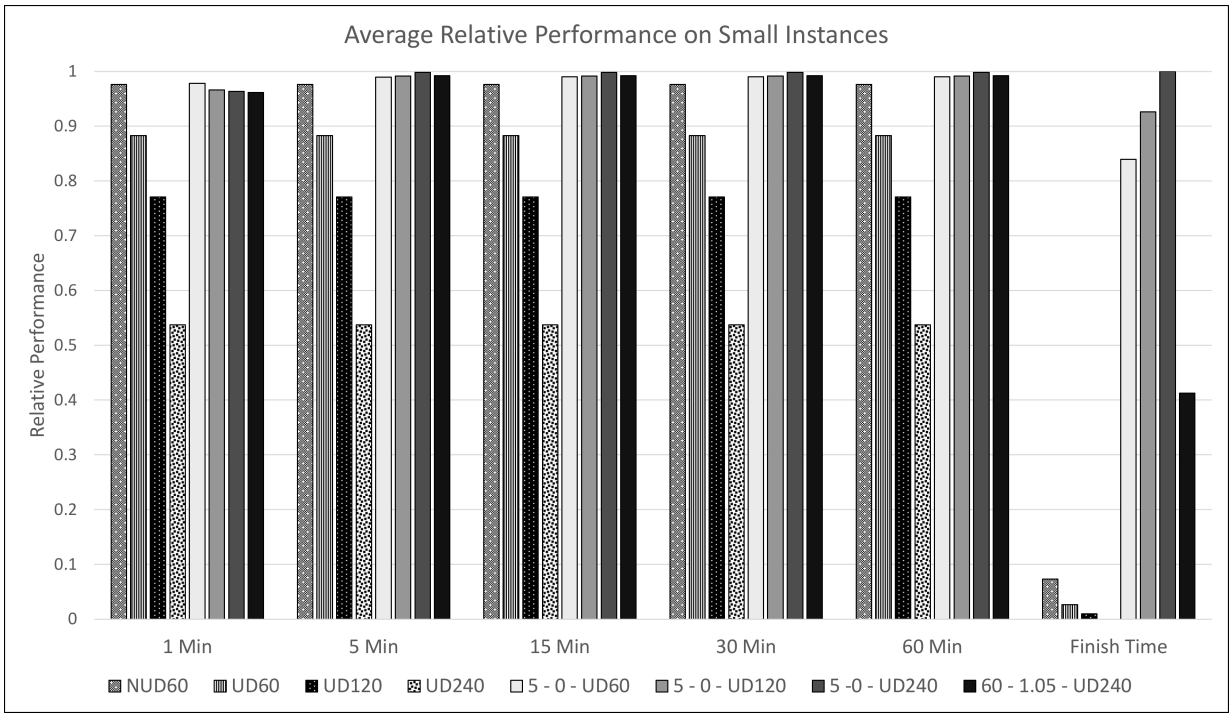


Figure 4.6: Results for small instances.

sets, compared to those with coarser timepoint sets. This demonstrates the weakness of using a fine discretization and the tradeoff between computational cost, solution quality, and discretization used. Furthermore, by comparing the coarser dynamic policies against their static counterparts, e.g. the 60 - 0 - UD240 dynamic policy and the UD240 static policy, after 1 minute has elapsed, we observe that even in cases where we desire solutions very quickly, and hence are forced to use coarse discretizations, there may still be a benefit to using an iterative approach. In this case, the 60 - 0 - UD240 dynamic policy is able to obtain approximately 77% of the greatest solution after 1 minute, while the UD240 obtains approximately 61% of the greatest solution on average.

We observe that all of the dynamic policies perform better than the static policies after 5 minutes have elapsed and that the NUD60 policy does not attain the same performance until 30 minutes have elapsed. Furthermore, at each checkpoint in time, a dynamic policy is performing best relative to the other static policies. This difference between the best performing static policy and the best performing dynamic policy ranges from 0% (at 30 or 60 minutes) - 20% (at 5 minutes). This demonstrates that for obtaining strictly the best performance, these policies were suitable on these medium sized problems. However, the same observation discussed in the previous paragraph still applies. As the problems are larger, the policies which use a more fine initial discretization must solve larger problems and hence we see that the 5 - 0 - UD60 policy does not attain similar performance to the other dynamic policies until the 15 minute checkpoint. This observation highlights the need to pick the parameters used for the dynamic policies suitably depending on the problem specifications, e.g. if solutions must be attained within 1 minute, then using a more coarse starting discretization would be preferable to starting with the UD60 discretization.

The tradeoff between the dynamic policies and the static policies in terms of time to completion is also less extreme than for the small instances. Most of the dynamic policies are still taking the longest to complete, and are taking longer than the NUD60 policy, but the amount of extra time spent is less significant than when the instances were small. Overall, over these instances, it appears that the 60 - 1.05 - UD240 policy is able to remain one of the best performers at each check point and also terminates sooner than the other dynamic policies and NUD60. This makes the 60 - 1.05 - UD240 an attractive alternative to the NUD60 static discretization over these problems.

Depending on the user's preferences, on medium size instances it appears to be beneficial to use a dynamic timepoint policy over a static discretization. However, this decision will be influenced by how large the problems to solve are, the amount of performance the user must obtain, and the time restrictions the problems must be solved within.

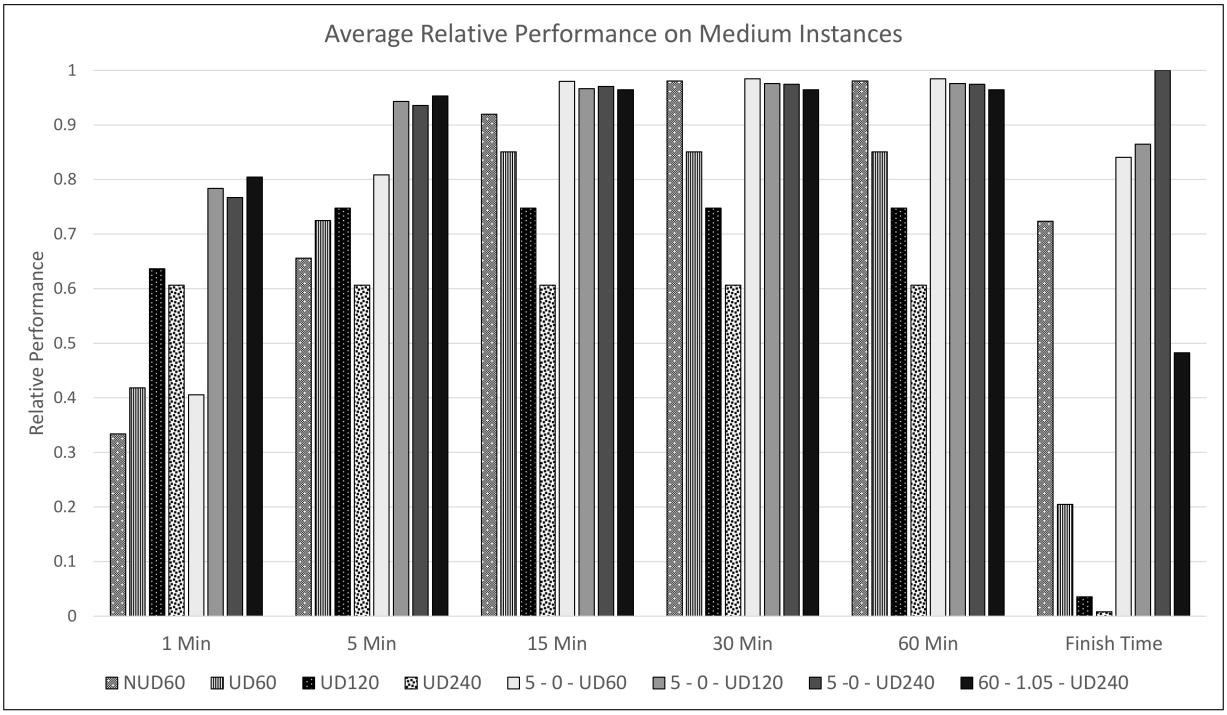


Figure 4.7: Results for medium instances.

4.3.5 Performance on Large Size Problems

Figure 4.8 shows the performance results on the large instances. It is at this point that we observe the breaking point of the NUD60 policy. On these instances, the NUD60 policy does not become competitive with the other policies until the 30 minute mark, and furthermore does not obtain a feasible solution within 1 minute on any of the large instances. The dynamic policies again offer good performance relative to the other policies at all checkpoints in time. The extra time cost associated with the dynamic policies is further reduced compared to the small and medium size instances. Over these instances, our iterative method offers a better alternative to the NUD60 discretization for users who require higher performance than a uniform discretization may offer by either being equal to or exceeding the performance of the NUD60 policy at each checkpoint and by terminating in less than 30% of the time. However, a coarse static policy may still be the preferable option if very fast solve times are the top priority, e.g. one needs solutions as fast as possible.

Furthermore, we see a benefit to increasing the amount of time to allow between solutions in these instances between the 60 - 1.05 - UD240 and the 5 - 0 - UD240 policies. It may be that the extra time we allow to spend searching for solutions with these large problems, leads to a better, more careful selection of timepoints to add to our timepoint sets. This observation highlights again the importance of the parameter selection discussed in section 4.3.1.

It is with the large size problems that we are able to best demonstrate where the iterative refinement method pulls ahead of the static discretizations. When the problems considered are sufficiently large, the NUD60 discretization produces many timepoints which makes the resulting problem very computationally expensive. By starting with a coarse discretization and then refining the set of timepoints iteratively, we are able to increase the solution quality, without incurring the same computational expense as starting with the NUD60 policy.

4.3.6 Per Iteration Analysis

In this section we analyze the behavior of the dynamic framework on the medium sized instances with respect to each iteration. We choose to discuss the medium size instances here because most of the instances tested fell into this category (see Table 4.6); however the observed trends are similar for the small and large sized instances. Note that for these medium sized instances, the maximum number of iterations that were performed by the

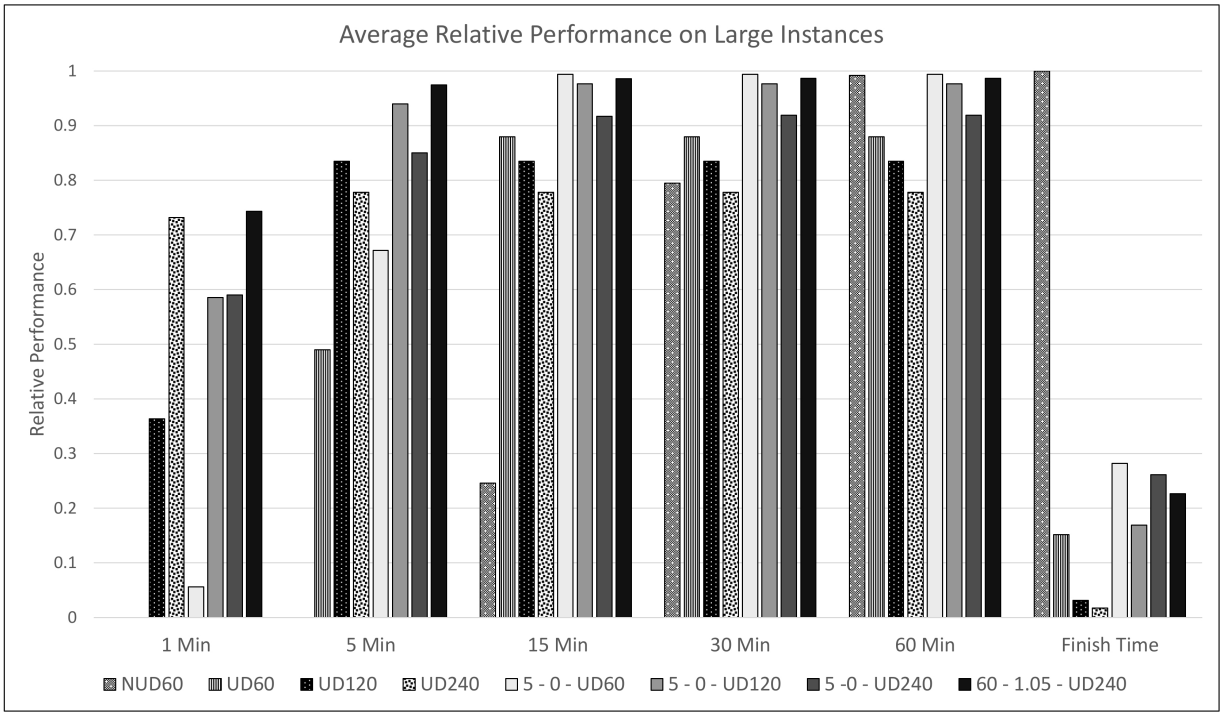


Figure 4.8: Results for large instances.

60 - 1.05 - UD240 policy was five, and therefore data for this policy does not show up in the results for iterations greater than five. Furthermore, because a given policy will not necessarily perform the same number of iterations on each problem instance (e.g. the 5 - 0.0 - UD120 policy may perform 3 iterations for medium sized instance 1, and perform 8 iterations for medium sized instance 2), the averages reported in this section are obtained using less data points for later iterations than earlier iterations.

The actual number of instances that were used for each policy and iteration number are presented in Table 4.7. The first column denotes the iteration number, and columns 2 - 4 denote how many instances were run for each policy such that the policy used that many iterations. For example, Table 4.7 shows that there were 28 instances where the 5 - 0.0 - UD60 policy took at least 3 iterations to complete, but only 22 instances where it took at least 4 iterations to complete. All policies terminated within 14 iterations over all medium sized instances. From Table 4.7 we are able to observe that the policies which start with more coarse uniform discretizations generally iterate more times before termination. This is not overly surprising as the more coarse policies start with fewer timepoints and therefore should be able to be solved more quickly. Since we impose the same `its_tl` time limit for each policy, then using a more coarse discretization should allow for more iterations. This does not hold for the 60 - 1.05 - UD240 policy however because of the 5% improvement that we impose between iterations. We observe that most instances complete in 3 iterations, with all of the instances completing within 5 iterations.

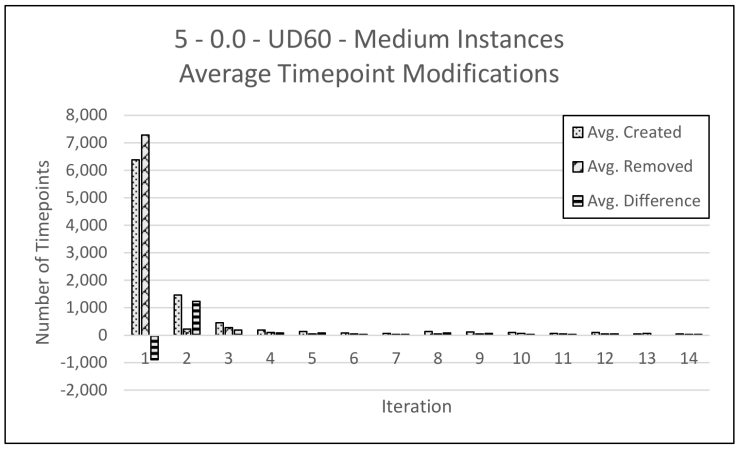
We now discuss the average number of timepoints that were added, removed, and their difference between iterations. This information is presented by Figure 4.9. We may immediately discern that regardless of the policy considered, after a few iterations there is a sharp decline in both the number of timepoints that are being added and the number of timepoints that are being removed between iterations. This trend shows that the timepoint sets undergo large changes at the beginning of the procedure and then become relatively stable after a few iterations. In other words, these results show that the iterative procedure converges to a single timepoint set rapidly. Moreover, we observe that there are a large number of timepoints which are removed in the beginning iterations. The number of timepoints removed is directly related to the granularity of the initial discretization, where more fine discretizations have more timepoints removed and in particular the 5 - 0.0 - UD60 policy removes more timepoints than it adds after the first iteration. This shows that using a uniform discretization will include many timepoints which are not required for obtaining high quality solutions. By observing the relative magnitudes of the timepoint changes between iterations, we see that the amount of timepoints modifications done by policies which start with more coarse discretizations decreases more slowly than policies which start with more fine discretizations. This is reasonable because the quality of the

Table 4.7: Number of instances considered for each iteration number.

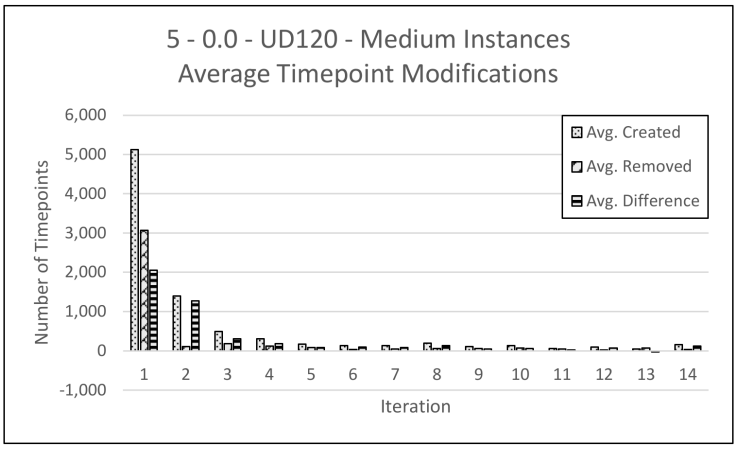
Iteration Number	5 - 0.0 - UD60 (# Instances)	5 - 0.0 - UD120 (# Instances)	5 - 0.0 - UD240 (# Instances)	60 - 1.0.5 - UD240 (# Instances)
1	34	34	34	34
2	34	34	34	34
3	28	33	34	34
4	22	27	34	10
5	21	23	31	2
6	18	23	28	0
7	13	17	21	0
8	9	11	15	0
9	7	9	12	0
10	4	5	10	0
11	3	4	9	0
12	3	4	4	0
13	3	3	3	0
14	3	2	2	0

initial solutions is directly related to the granularity of the starting discretization. Starting with a fine discretization allows us to make many timepoint modifications initially but then quickly converge to a suitable set of timepoints.

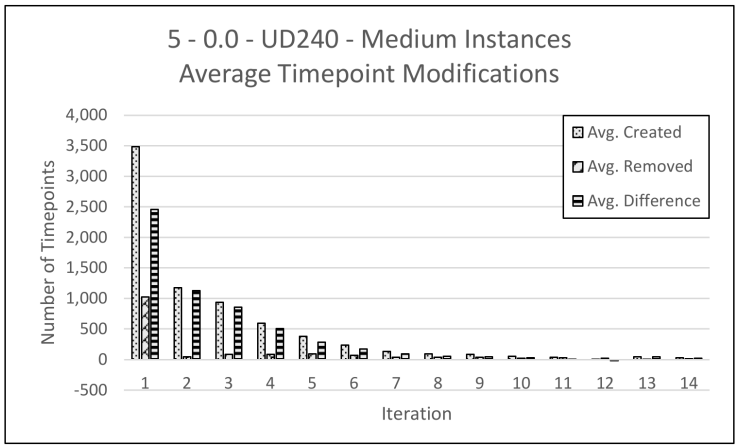
We now analyze the average number of timepoints considered for each iteration. These results are presented by Figure 4.10. Based on the discussion on the average number of timepoint modifications above, the results of Figure 4.10 are expected. In the first iteration, we are able to observe the differences in the number of timepoints for the starting discretizations considered, UD60, UD120, and UD240. After the first iteration, we see a moderate increase in the number of timepoints for the 5 - 0.0 - UD120, 5 - 0.0 - UD240, and 60 - 1.05 - UD240 discretizations, while there is a decline in the number of timepoints for policy 5 - 0.0 - UD60, as was depicted by Figure 4.9a. These observations support the idea that policies which start with coarse discretizations are able to identify many new timepoints which may be beneficial to a schedule formulation, whereas the policies which start with fine discretizations are not able to identify as many timepoints to add because many are already included. Note that the average number of timepoints considered by the NUD60 static policy for small, medium, and large instance sizes are as follows: small instances - 18,117, medium instances - 45,763, large instances - 92,384. Comparing the



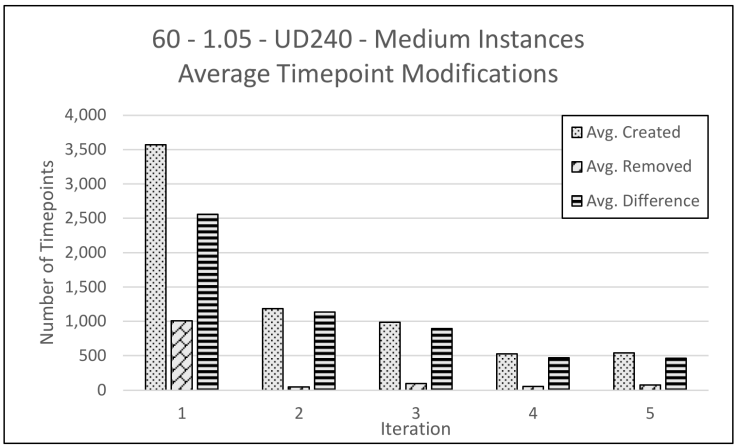
(a) Timepoint modifications for 5 - 0.0 - UD60 policy.



(b) Timepoint modifications for 5 - 0.0 - UD120 policy.



(c) Timepoint modifications for 5 - 0.0 - UD240 policy.



(d) Timepoint modifications for 60 - 1.05 - UD240 policy.

Figure 4.9: The average number of timepoints added, removed, and their difference between iterations over each policy.

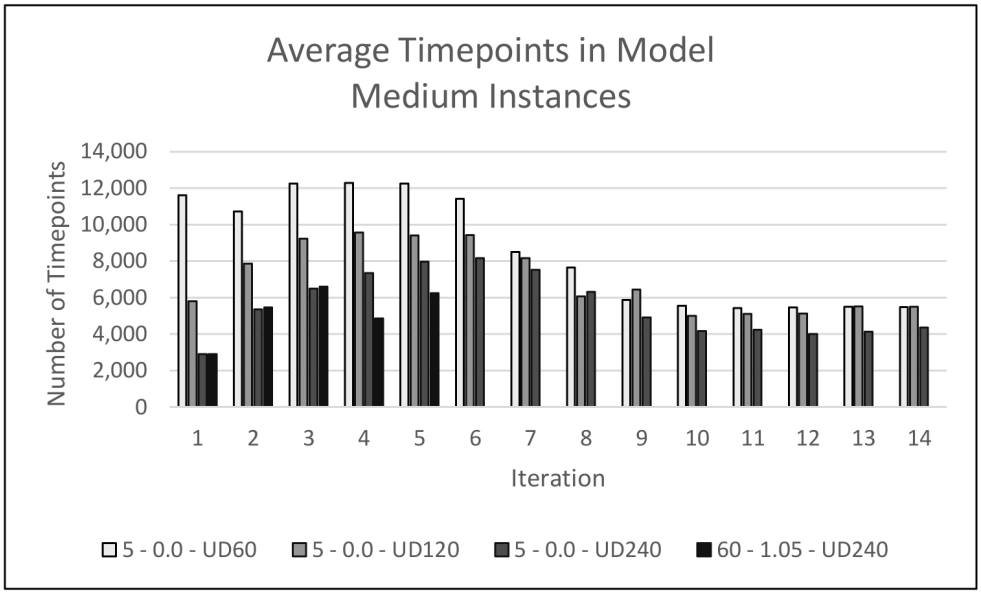


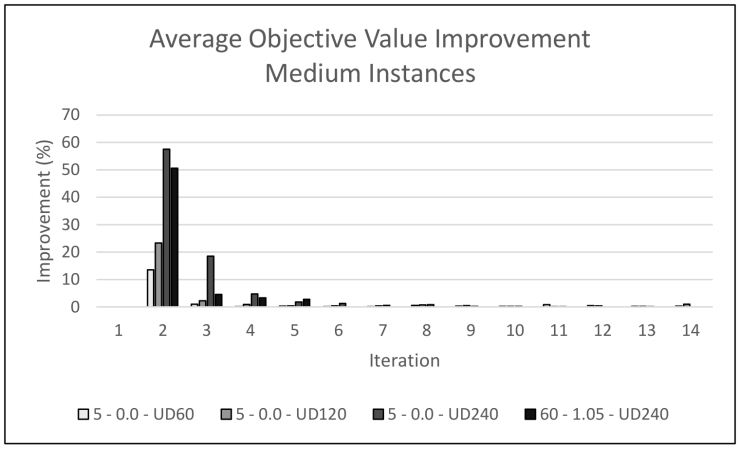
Figure 4.10: Per iteration timepoint differences for medium sized instances.

number of timepoints the NUD60 policy used on the medium sized instance to the number of timepoints the dynamic policies used shows more than a 75% reduction in the number of timepoints considered in some cases. This reduction in timepoints translates into a reduction in problem size as well as shown in Table 4.8. This table presents the average problem size (in terms of number of variables and constraints) for each policy and iteration and compares it to the average problem size of the NUD60 discretization. The maximal problem sizes among all iterations for each policy have been boldfaced. Therefore, reducing the number of timepoints may be useful in cases where the size of the problems considered surpasses the capabilities of the available hardware (e.g. CPU or memory limitations). Instead of using a fine static discretization, a user may iteratively refine the time grids by solving smaller sized problems, yet still achieve good solution quality as discussed in the earlier results sections 4.3.3 - 4.3.5.

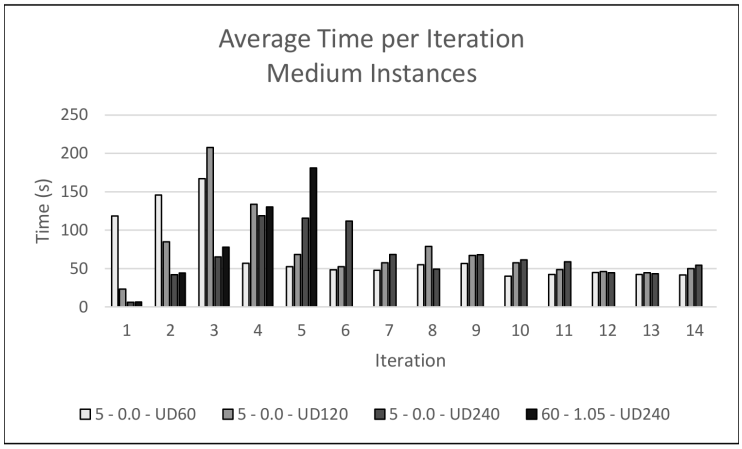
Figure 4.11a presents the per iteration performance improvements for each policy, in terms of objective value percentage increase. Similar to the results discussed above when considering timepoint differences between iterations, most of the performance improvements are obtained within the first few iterations of the framework. This is unsurprising as we have already noted that the timepoint sets are only marginally changed in the later iterations so we do not expect to gain as much improvement then. These results suggest that only a few iterations are necessary and that perhaps a stopping criteria based on

Table 4.8: Average number of variables and constraints per iteration, numbers reported in 1,000's.

Iteration Number	5 - 0.0 - UD60		5 - 0.0 - UD120		5 - 0.0 - UD240		60 - 1.05 - UD240		NUD60	
	Vars	Cons	Vars	Cons	Vars	Cons	Vars	Cons	Vars	Cons
1	329	192	171	101	93	56	93	56	**1,611**	**900**
2	1,191	621	1,002	521	796	412	802	416		
3	1,250	654	1,121	583	928	481	935	485		
4	1,271	664	1,122	584	1,034	536	**1,108**	**569**		
5	1,268	663	1,129	588	1,088	564	951	493		
6	1,304	680	1,156	601	1,106	573	N/A	N/A		
7	1,348	696	1,213	628	1,086	562	N/A	N/A		
8	1.428	735	**1,431**	**734**	1,170	603	N/A	N/A		
9	**1,442**	**739**	1,369	704	1,300	666	N/A	N/A		
10	1,156	595	1,165	598	**1,396**	**713**	N/A	N/A		
12	1,179	606	1,131	581	1,114	570	N/A	N/A		
11	1,176	604	1,133	582	1,387	709	N/A	N/A		
13	1,182	608	1,176	605	1,133	580	N/A	N/A		
14	1,175	604	1,141	587	1,109	569	N/A	N/A		



(a) Per iteration objective value improvements.



(b) Time taken per iteration.

Figure 4.11: Average objective value improvements and time taken per iteration.

the number of iterations carried out may allow us to obtain most of the benefits of the procedure, without incurring all of the computational cost.

Figure 4.11b shows the time taken to complete each iteration for each policy. From this figure, we observe an initial increase in the solving time per iteration. This increase corresponds to the rapidly changing timepoint sets which render previously obtained solutions not as relevant a starting solution as for later iterations where timepoints do not change much. After the initial period of increasing solution times, we see that the time per iteration begins to decrease and stabilize to approximately 50 seconds. By limiting the number of iterations as suggested previously, we may remove the extra time spent in the later iterations without much improvement.

4.4 Chapter Summary

In this chapter, we presented our proposed iterative method for timepoint refinement. We presented our problem as a more general scheduling problem over a time layered graph and explained how our proposed heuristics are designed to reduce the amount of time between when material arrives at a state and when it is able to exit. We presented results comparing our dynamic method to commonly used discretizations and showed that in our tests, there were benefits in terms of both computational time and solution quality compared to the static discretizations, when problems are sufficiently large. When problems are small, we

showed that our method is also able to provide high quality solutions, but that the increase in time for our method to terminate compared to using static discretizations may be large. We discussed the importance of choosing parameters for our dynamic method and how the choice of parameters plays a role in how the method performs. Furthermore, we analyzed the evolution of the iterative framework on a per iteration basis and showed that most of the improvements and timepoint modifications are done in the first few iterations.

Chapter 5

Conclusions

The goal of this thesis was to investigate two questions pertaining to scheduling multipurpose facilities: when should rescheduling be done and how to efficiently solve the resulting scheduling problems? The aim of the first study was to better understand the role facility parameters play when determining how often to reschedule a multipurpose facility. The second was to present and validate an iterative method for choosing time grids for general scheduling problems over time layered graphs.

The work in Chapter 3 presents a comparison between several different periodic rescheduling policies varying from generating a schedule four times per day to generating a schedule only once every five days. Experiments were conducted simulating a real analytical services facility to compare the performance between the policies using a rolling horizon routine and the non uniform discrete time model presented in [22]. The effectiveness of each policy was measured throughout the experiments using percentage of jobs completed, average job makespan, and proportion of jobs on time as performance measures.

Based on the results obtained through the computational experiments, this study shows that choosing a suitable rescheduling policy can depend greatly on the environment of the facility that is being modeled. By varying the capacity of the processes in the experiments we observed that in some cases, less frequent rescheduling policies may outperform more frequent rescheduling policies both in job completion and proportion of jobs on time. In particular, in some experiments, we were able to observe nearly a complete reversal of the results obtained when nominal parameter values were used.

When nominal parameter values were used, we observed that more frequent rescheduling can have a significant positive impact on improving the proportion of jobs on time and makespan of a production plant.

In environments where capacity is not much of a concern or where very short lead times are required, more frequent rescheduling policies seem to be best. However, if long lead times are acceptable or capacity is the main limiting constraint, then scheduling less frequently with longer horizons can be beneficial.

Chapter 4 presented the study done on iteratively refining time grids. The general framework used for our method was presented along with a discussion on the motivating ideas used to choose new timepoints. Computational experiments were performed which showed that by refining the time grids of our scheduling problem with the proposed framework, we were able to improve performance over the conventionally used discretizations without substantially increasing the computational cost.

In particular, the NUD60 discretization which has performed well on small problems in practice began to perform worse as problem sizes get large. However, the performance of the iterative policies stayed relatively stable across various problem sizes, but with improved tradeoff as problems became large. Moreover, analyzing the performance of the framework over each iteration showed that most of the performance improvements and timepoint modifications are done in the first few iterations.

The results of these experiments show that using a refinement strategy can improve solution quality over the conventionally used uniform discretizations even when very short solving times are required. It is also worth noting that the proposed method is a general strategy which may be applied to other scheduling applications with similar requirements, bypassing the need to experimentally determine efficient time grids.

5.1 Future Work

There are a number of ways that the work presented in this thesis may be extended. With respect to rescheduling frequency, future work could consider implementing further sources of uncertainty such as job retesting if a process malfunctions or the possibility of machine breakdowns. With these augmentations, a more robust scheduling policy than fixed periodic may be desirable to reschedule immediately when a more serious disruptive event occurs and to forgo rescheduling when it is not necessary. Additionally, more research into quantifying the costs of rescheduling such as schedule disruption and latency introduced by deviating from the originally intended schedule could be beneficial. In this study we assumed that these costs were negligible but a study focusing on quantifying and measuring these costs could help us to better understand the other side effects of rescheduling. Finally, an investigation into how the results vary with different job arrival patterns, such as bursty arrivals, may also be an interesting contribution.

On the topic of choosing time grids for scheduling problems, there are several directions one could explore. Investigations into selecting a more specialized initial time grid rather than simply using a uniform discrete variant could help the iterative method converge even more quickly. Additional heuristics for choosing which timepoints to add or remove during the method could be proposed and their efficacy tested. This could be especially useful when the performance differences between iterations become small. Furthermore, there are a number of parameter choices that are used for the iterative method, and the choices for these parameters can have a large impact on the efficiency of the policies. More research into how to fine tune these parameters, particularly for different problem applications, would be a worthwhile endeavor. The values for these parameters were chosen experimentally in this work, but an algorithmic method of selecting these values would be helpful when applying this method to other applications or facilities.

References

[1] J. Andersen, M. Christiansen, Crainic T. G., and R. Grønhaug. Branch and price for service network design with asset management constraints. *Transportation Science*, 45(1):33–49, 2011.

[2] A. Baykasoğlu and F. S. Karaslan. Solving comprehensive dynamic job shop scheduling problem by using a grasp-based approach. *International Journal of Production Research*, 55(11):3308–3325, 2017.

[3] N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh. The continuous-time service network design problem. *Operations Research*, 65(5):1303–1321, 2017.

[4] I. A. Chaudhry and A. A. Khan. A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23:551–591, 2016.

[5] L. K. Church and R. Uzsoy. Analysis of periodic and event-driven rescheduling policies in dynamic shops. *Int. J. Computer Integrated Manufacturing*, 5(3):153–163, 1992.

[6] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer Programming*. Springer, 2014.

[7] IBM ILOG CPLEX. Cplex user's manual, 2018.

[8] T. G. Crainic, M. Hewitt, M. Toulouse, and D. M. Vu. Service network design with resource constraints. *Transportation Science*, 50(4):1380–1393, 2016.

[9] E. D. Dolan and J. J. More. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[10] I. Dunning, J. Huchette, and M. Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[11] A. Erera, M. Hewitt, M. Savelsbergh, and Y. Zhang. Improved load plan design through integer programming based local search. *Transportation Science*, 47(3):412–427, 2013.

[12] J. Fang and Y. Xi. A rolling horizon job shop rescheduling strategy in the dynamic environment. *Int J Adv Manuf Technol*, 13(3):227–232, 1997.

[13] C. Floudas and X. Lin. Continuous-time versus discrete-time approaches for scheduling of chemical processes: a review. *Computers and Chemical Engineering*, 28(11):2109–2129, 2004.

[14] A. Froger, M. Gendreau, J. E. Mendoza, E. Pinson, and L.-M. Rousseau. Maintenance scheduling in the electricity industry: A literature review. *European Journal of Operational Research*, 251:695–706, 2016.

[15] D. Gupta and C. Maravelias. On deterministic online scheduling: Major considerations, paradoxes and remedies. *Computers and Chemical Engineering*, 94(2):312–330, 2016.

[16] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2018.

[17] M. Heydari and A. Soudi. Predictive/reactive planning and scheduling of a surgical suite with emergency patient arrival. *J Med Syst*, 40:1–9, 2016.

[18] K. Hozak and J. A. Hill. Issues and opportunities regarding replanning and rescheduling frequencies. *International Journal of Production Research*, 47(18):4955–4970, 2009.

[19] M. H. Kim and Y.-D. Kim. Simulation-based real-time scheduling in a flexible manufacturing system. *Journal of Manufacturing Systems*, 13(2):85–93, 1994.

[20] R. Koller, L. A. Ricardez-Sandoval, and L. Biegler. Stochasic back-off algorithm for simultaneous design, control, and scheduling of multiproduct systems under uncertainty. *AIChE J*, 64(7):2379–2389, 2018.

[21] E. Kondili, C. C. Pantelides, and R. W. H. Sargent. A general algorithm for short-term scheduling of batch operations—i. milp formulation. *Computers and Chemical Engineering*, 17(2):211–227, 1993.

[22] S. Lagzi, D. Yeon Lee, R. Fukasawa, and L. A. Ricardez-Sandoval. A computational study of continuous and discrete time formulations for a class of short-term scheduling problems for multipurpose plants. *Ind. Eng. Chem. Res.*, 56(31):8940–8953, 2017.

[23] J. Lange and F. Werner. Approaches to modeling train scheduling problems as job-shop problems with blocking constraints. *Journal of Scheduling*, 21:191–207, 2018.

[24] H. Lee and C. T. Maravelias. Combining the advantages of discrete- and continuous-time scheduling models: Part 1. framework and mathematical formulations. *Computers and Chemical Engineering*, 116:176–190, 2017.

[25] R.-K. Li, Y.-T. Shyu, and S. Adiga. A heuristic rescheduling algorithm for computer-based production scheduling systems. *Int. J. Prod. Res.*, 31(8):1815–1826, 1993.

[26] A. F. Merchan, H. Lee, and C. T. Maravelias. Discrete-time mixed-integer programming models and solution methods for production scheduling in multistage facilities. *Computers and Chemical Engineering*, 94:387–410, 2016.

[27] L. Mockus and G. V. Reklaitis. Continuous time representation approach to batch and continuous process scheduling. 1. minlp formulation. *Ind. Eng. Chem. Res.*, 38(1):197–203, 1999.

[28] A. P. Muhlemann, A. G. Lockett, and C.-K. Farn. Job shop scheduling heuristics and frequency of scheduling. *International Journal of Production Research*, 29(2):227–241, 1982.

[29] D. C. Paraskevopoulos, G. Laporte, P. P. Repoussis, and C. D. Tarantilis. Resource constrained routing and scheduling: Review and research prospects. *European Journal of Operational Research*, 263:737–754, 2017.

[30] B. Patil, R. Fukasawa, and L. A. Ricardez-Sandoval. Scheduling of operations in a large-scale scientific services facility via multicommodity flow and an optimization-based algorithm. *Ind. Eng. Chem. Res.*, 54:1628–1639, 2015.

[31] M. E. Pfund and J. W. Fowler. Extending the boundaries between scheduling and dispatching: hedging and rescheduling techniques. *International Journal of Production Research*, 55(11):3294–3307, 2017.

[32] M. Pinedo. *Scheduling*. Springer, 2016.

[33] R. Renaldi and D. Friedrich. Multiple time grids in operational optimisation of energy systems with short- and long-term thermal energy storage. *Energy*, 133:784–795, 2017.

[34] I. Sabuncuoglu and S. Karabuk. Rescheduling frequency in an fms with uncertain processing times and unreliable machines. *Journal of Manufacturing Systems*, 18(4):268–283, 1999.

[35] I. Sabuncuoglu and O. B. Kizilisik. Reactive scheduling in a dynamic and stochastic fms environment. *International Journal of Production Research*, 41(17):4211–4231, 2003.

[36] G. Schilling and C. C. Pantelides. A simple continuous-time process scheduling formulation and a novel solution algorithm. *Computers and Chemical Engineering*, 20(2):S1221–S1226, 1996.

[37] R. Shafaei and P. Brunn. Workshop scheduling using practical (inaccurate) data part 1: The performance of heuristic scheduling rules in a dynamic job shop environment using a rolling time horizon approach. *International Journal of Production Research*, 37(17):3913–3925, 1999.

[38] N. Shah, C. C. Pantelides, and R. W. H. Sargent. A general algorithm for short-term scheduling of batch operations—ii. computational issues. *Computers and Chemical Engineering*, 17(2):229–244, 1993.

[39] T. Stablein and K. Aoki. Planning and scheduling in the automotive industry: A comparison of industrial practice at german and japanese makers. *Int. J. Production Economics*, 162:258–272, 2015.

[40] M. Steinrücke. Integrated production, distribution and scheduling in the aluminium industry: a continuous-time milp model and decomposition method. *International Journal of Production Research*, 53(19):5912–5930, 2015.

[41] K. Stuart and E. Kozan. Reactive scheduling model for the operating theatre. *Flex Serv Manuf J*, 24:400–421, 2012.

[42] A. Sundaramoorthy and C. T. Maravelias. Computational study of network-based mixed-integer programming approaches for chemical production scheduling. *Ind. Eng. Chem. Res.*, 50(9):5023–5040, 2011.

[43] M. A. H. van Elzakker, E. Zondervan, N. B. Raikar, I. E. Grossmann, and P. M. M. Bongers. Scheduling in the fmcg industry: An industrial case study. *Ind. Eng. Chem. Res.*, 51:7800–7815, 2012.

[44] S. Velez and C. Maravelias. Multiple and nonuniform time grids in discrete-time mip models for chemical production scheduling. *Computers and Chemical Engineering*, 53:70–85, 2013.

[45] S. Velez and C. T. Maravelias. Theoretical framework for formulating mip scheduling models with multiple and non-uniform discrete-time grids. *Computers and Chemical Engineering*, 72:233–254, 2015.

[46] S. Velez, A. F. Merchan, and C. T. Maravelias. On the solution of large-scale mixed integer programming scheduling models. *Chemical Engineering Science*, 136:139–157, 2015.

[47] G. E. Vieira, J. W. Herrmann, and E. Lin. Analytical models to predict the performance of a single-machine system under periodic and event-driven rescheduling strategies. *International Journal of Production Research*, 38(8):1899–1915, 2000.

[48] G. E. Vieira, J. W. Herrmann, and E. Lin. Predicting the performance of rescheduling strategies for parallel machine systems. *Journal of Manufacturing Systems*, 19(4):256–266, 2000.

[49] G. E. Vieira, J. W. Herrmann, and E. Lin. Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling*, 6(1):39–62, 2003.

[50] B. Wang, X. Han, X. Zhang, and S. Zhang. Predictive-reactive scheduling for single surgical suite subject to random emergency surgery. *J Comb Optim*, 30:949–966, 2015.

[51] Y. Wang, Z. Liao, T. Tang, and B. Ning. Train scheduling and circulation planning in urban rail transit lines. *Control Engineering Practice*, 61:112–123, 2017.

[52] C. A. Yano and R. C. Carlson. Interaction between frequency of rescheduling and the role of safety stock in material requirements for planning systems. *International Journal of Production Research*, 25(2):221–232, 1987.

[53] X. Zhang and R. W. H. Sargent. The optimal operation of mixed production facilities—a general formulation and some approaches for the solution. *Computers and Chemical Engineering*, 20(6-7):897–904, 1996.

APPENDICES

Appendix A

Facility Parameter Values

Table A.1 below includes the normalized process parameter values used for our experiments. The first column is the process index, the second column includes the capacity of each process resource as a fraction of the largest resource capacity among all processes. The third presents the processing time of each process as a portion of the greatest processing time over all processes. The fourth column includes the number of resources available for each process.

Table A.1: Normalized process parameters used during experiments.

Process ID (u)	Process Capacity ($\kappa(u)$)	Processing Time ($\tau(u)$)	Number of Resources ($\rho(u)$)
1	0.724638	0.001488	2
2	0.043478	0.005952	3
3	0.039855	0.047619	1
4	0.000725	0.142857	4500
5	0.362319	0.037202	4
6	0.036232	0.018651	10
7	0.036232	0.099206	2
8	0.036232	0.018651	6
9	0.036232	0.019841	2
10	0.002174	0.000794	1
11	0.036232	0.019841	3

Continued on next page

Table A.1 – *Continued from previous page*

Process ID (u)	Process Capacity ($\kappa(u)$)	Processing Time ($\tau(u)$)	Number of Resources ($\rho(u)$)
12	0.036232	0.014881	1
13	0.036232	0.00496	3
14	0.07971	0.107143	1
15	0.036232	0.009921	1
16	0.030435	0.003968	2
17	0.030435	0.003968	2
18	0.021739	0.10119	4
19	0.156522	0.029762	2
20	0.015217	0.014881	2
21	0.005797	0.018849	1
22	0.034783	0.061012	1
23	0.000725	0.016567	1
24	0.005072	0.142857	1
25	0.042754	0.209425	1
26	0.028986	0.165179	2
27	0.028986	0.744048	40
28	0.021739	0.035714	2
29	0.005797	0.053571	1
30	0.03913	0.050595	4
31	0.021739	0.005952	1
32	0.019565	0.011409	1
33	0.019565	0.040675	1
34	0.018841	0.013889	1
35	0.018841	0.013889	1
36	0.018841	0.013889	1
37	0.018116	0.0125	1
38	0.017391	0.013889	1
39	0.019565	0.010913	1
40	0.144928	0.011905	1
41	0.073913	0.011905	2
42	0.05942	0.000992	2

Continued on next page

Table A.1 – *Continued from previous page*

Process ID (u)	Process Capacity ($\kappa(u)$)	Processing Time ($\tau(u)$)	Number of Resources ($\rho(u)$)
43	0.221014	0.011905	2
44	0.108696	0.02381	1
45	0.074638	0.005952	1
46	0.09058	0.017857	1
47	0.09058	0.017857	1
48	0.008696	0.005952	1
49	0.042754	0.008929	1
50	0.014493	0.001984	1
51	0.043478	0.005952	1
52	0.004348	0.005952	2
53	0.028986	0.005952	1
54	0.043478	0.005952	1
55	0.021739	0.005952	2
56	0.347826	0.017857	3
57	0.318841	0.02381	2
58	0.072464	0.041667	1
59	0.036232	0.098214	1
60	0.008696	0.013393	2
61	0.021739	0.002976	1
62	0.005797	0.026786	3
63	0.156522	0.011905	4
64	0.313043	0.021825	2
65	0.318841	0.021825	4
66	0.021739	0.005952	5
67	0.052174	0.014385	1
68	0.049275	0.02629	1
69	0.037681	0.048611	1
70	0.021739	0.066964	1
71	0.000725	0.024306	2
72	0.013043	0.048115	1
73	0.001449	0.001984	2

Continued on next page

Table A.1 – *Continued from previous page*

Process ID (u)	Process Capacity ($\kappa(u)$)	Processing Time ($\tau(u)$)	Number of Resources ($\rho(u)$)
74	0.043478	0.005952	1
75	0.030435	0.006448	2
76	0.030435	0.000992	2
77	0.06087	0.005952	1
78	0.004348	0.032738	2
79	0.030435	0.004167	2
80	0.013043	0.001488	2
81	0.030435	0.001488	2
82	0.030435	0.007937	2
83	0.013043	0.007937	2
84	0.030435	0.004167	1
85	0.030435	0.004167	2
86	0.010145	0.047619	1
87	0.021739	0.006746	1
88	0.092754	0.073016	4
89	0.092754	0.006349	2
90	0.000725	0.000992	1
91	0.000725	0.142857	1
92	0.108696	0.142857	2
93	0.018841	0.012897	1
94	0.018841	0.012897	1
95	0.018116	0.013988	1
96	0.018841	0.011905	1
97	0.019565	0.042659	1
98	0.130435	0.03869	1
99	0.173913	0.142857	1
100	0.082609	0.31498	1
101	0.084058	0.01746	1
102	0.521739	0.072917	10
103	0.347826	0.046726	10
104	0.521739	0.070536	10

Continued on next page

Table A.1 – *Continued from previous page*

Process ID (u)	Process Capacity ($\kappa(u)$)	Processing Time ($\tau(u)$)	Number of Resources ($\rho(u)$)
105	0.121739	0.017857	10
106	0.217391	0.024802	10
107	1.0	0.138393	10
108	0.26087	0.025595	2
109	0.081159	0.124603	8
110	0.097826	0.113194	8
111	0.05942	0.025198	8
112	0.05942	0.142659	8
113	0.053623	0.127083	8
114	0.374638	0.159226	8
115	0.195652	0.077976	8
116	0.191304	0.133929	1
117	0.15	0.136905	5
118	0.007971	0.000298	6
119	0.021739	0.000992	1
120	0.021739	0.000992	1
121	0.007971	0.002976	1
122	0.007971	0.000694	1
123	0.347826	0.047619	1
124	0.386232	0.047619	1
125	0.104348	0.047619	1
126	0.26087	1.0	1
127	0.521739	1.0	1
128	0.072464	0.041667	1
129	0.065217	0.029762	2
130	0.03913	0.012897	1
131	0.008696	0.125	1
132	0.03913	0.102183	1
133	0.03913	0.012897	1
134	0.011594	0.02629	1
135	0.011594	0.029762	1

Continued on next page

Table A.1 – *Continued from previous page*

Process ID (u)	Process Capacity ($\kappa(u)$)	Processing Time ($\tau(u)$)	Number of Resources ($\rho(u)$)
136	0.03913	0.019841	1
137	0.018841	0.075496	1
138	0.03913	0.007937	2
139	0.018841	0.030556	1
140	0.03913	0.008929	4
141	0.069565	0.064782	3
142	0.069565	0.006944	4
143	0.104348	0.020536	1
144	0.073913	0.091071	1
145	0.073913	0.040476	1
146	0.073913	0.040476	1
147	0.043478	0.005952	1
148	0.010145	0.166667	2
149	0.010145	0.044643	2
150	0.005797	0.041667	2
151	0.005797	0.018353	1
152	0.005797	0.001488	2
153	0.043478	0.002976	2
154	0.015942	0.00248	2
155	0.005797	0.003175	1
156	0.000725	0.000198	1
157	0.013043	0.029762	1
158	0.019565	0.113095	1
159	0.015942	0.005952	1
160	0.014493	0.089286	1
161	0.000725	0.001587	1
162	0.000725	9.9e-5	1
163	0.043478	0.105655	1
164	0.043478	0.105655	1
165	0.043478	0.105655	1
166	0.318841	0.160714	4

Continued on next page

Table A.1 – *Continued from previous page*

Process ID (u)	Process Capacity ($\kappa(u)$)	Processing Time ($\tau(u)$)	Number of Resources ($\rho(u)$)
167	0.318841	0.166667	4
168	0.017391	0.056548	1
169	0.011594	0.15873	1
170	0.017391	0.02381	1
171	0.004348	0.03869	1
172	0.014493	0.006151	1
173	0.008696	0.012401	1
174	0.023188	0.008929	1
175	0.000725	9.9e-5	6
176	0.000725	9.9e-5	1
177	0.01087	0.001488	1
178	0.000725	9.9e-5	1
179	0.000725	0.000893	1
180	0.000725	9.9e-5	1
181	0.000725	0.000992	3
182	0.008696	0.016667	1
183	0.000725	0.001488	1
184	0.000725	9.9e-5	1
185	0.004348	0.002183	5
186	0.000725	9.9e-5	2
187	0.724638	0.000992	1
188	0.724638	0.142857	1
189	0.724638	0.142857	1

Appendix B

Additional Rescheduling Performance Profiles

Figures B.1 - B.22 present performance profiles for the experiments conducted in Section 3.5. Figures B.1 - B.3, present the performance profiles for test instances using moderate facility load. Figures B.4 - B.15, present jobs on time performance profiles for test instances with various facility loads. Figures B.16 - B.19, present job completion performance profiles for test instances with 1/2x, 2x, 4x, 8x capacity. Figures B.20 - B.22, present jobs on time performance profiles for test instances with 1/8x, 1/4x capacity.

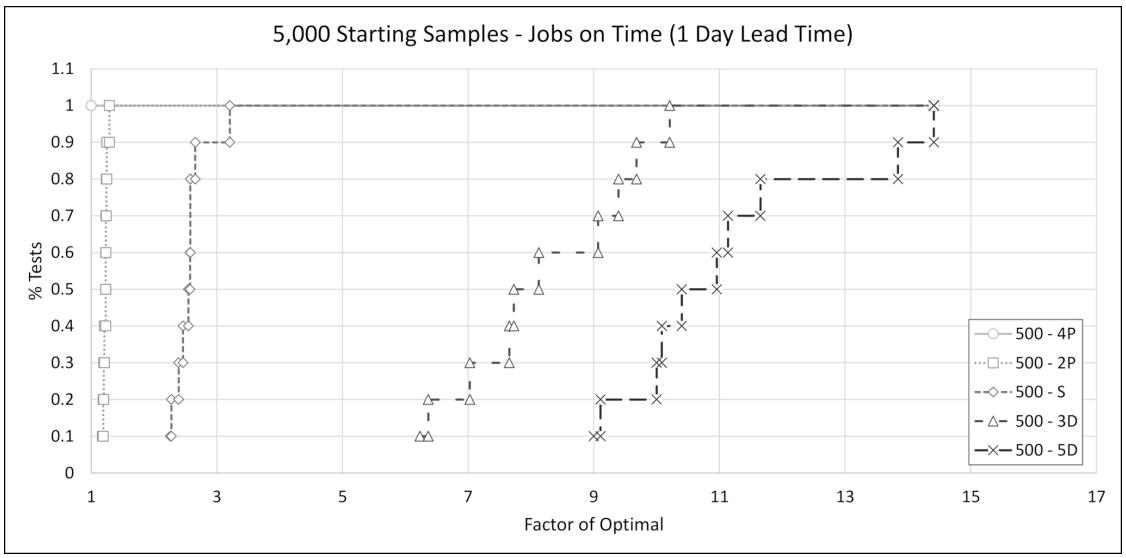


Figure B.1: Proportion of jobs on time for one day lead times and instances starting with 5,000 samples and 500 samples arriving each day.

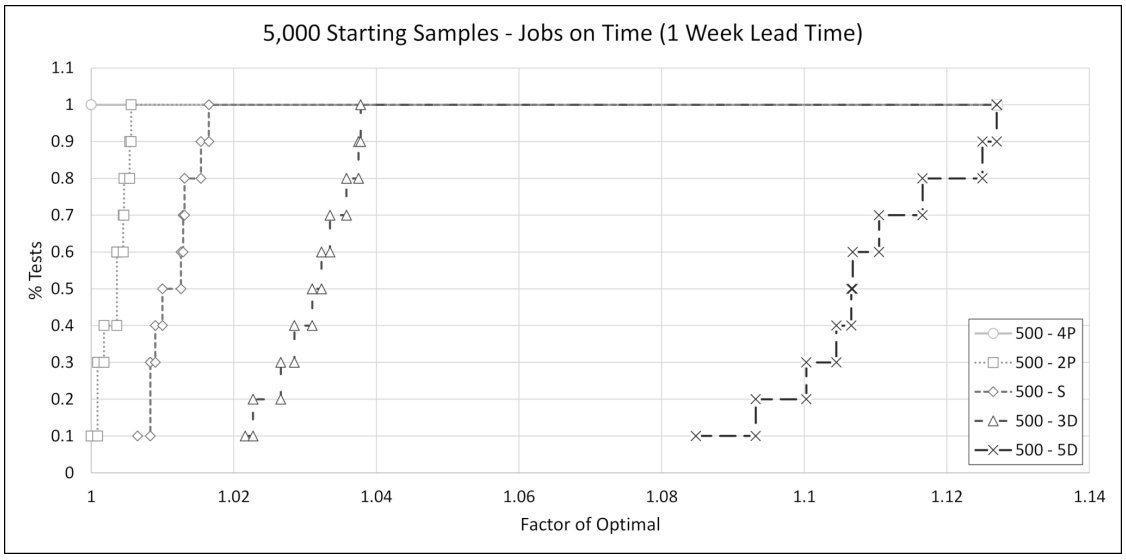


Figure B.2: Proportion of jobs on time for one week lead times and instances starting with 5,000 samples and 500 samples arriving each day.

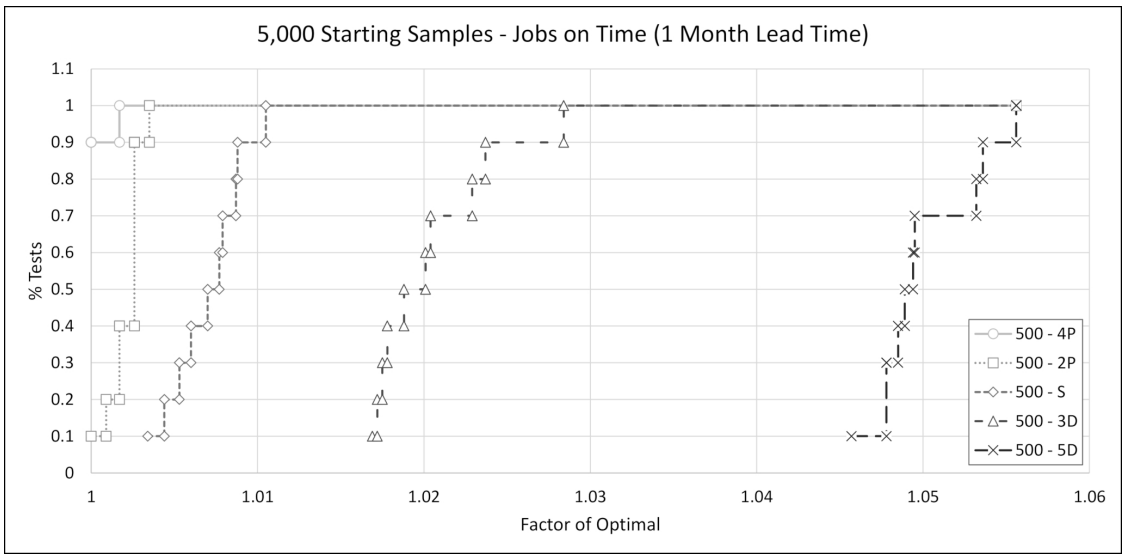


Figure B.3: Proportion of jobs on time for one month lead times and instances starting with 5,000 samples and 500 samples arriving each day.

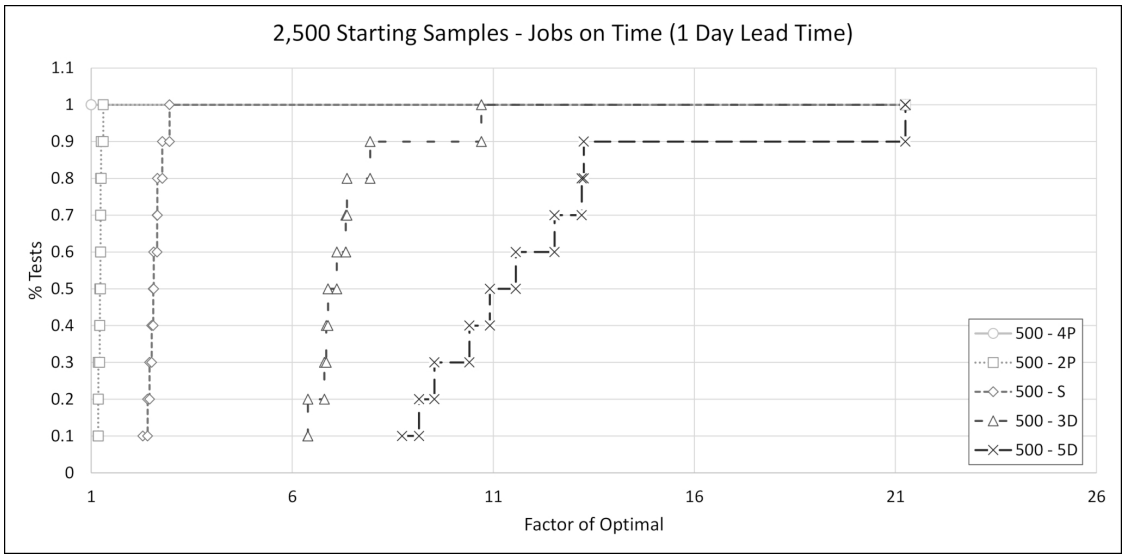


Figure B.4: Proportion of jobs on time for one day lead times and instances starting with 2,500 samples and 500 samples arriving each day.

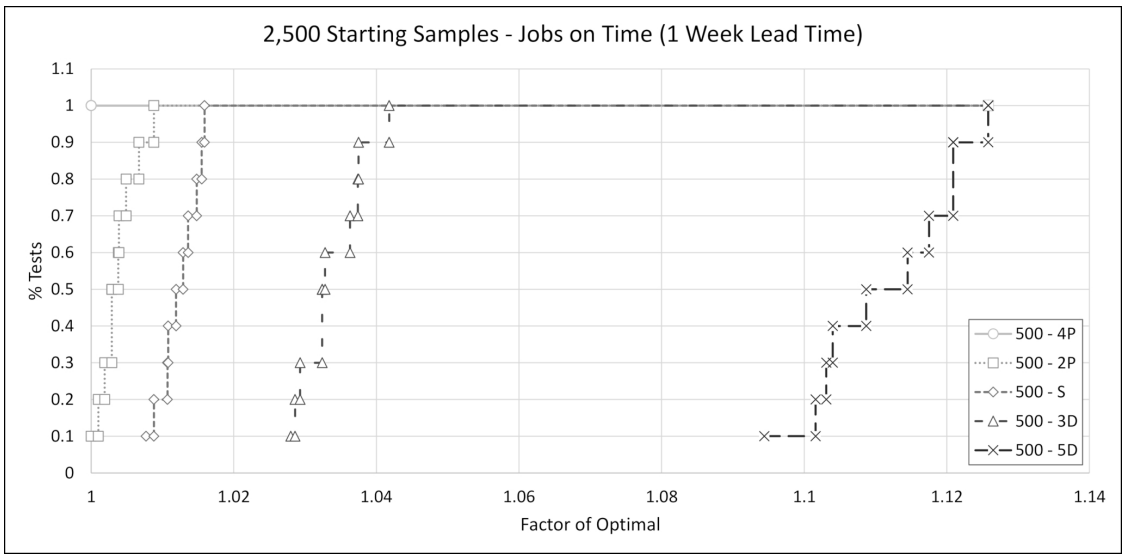


Figure B.5: Proportion of jobs on time for one week lead times and instances starting with 2,500 samples and 500 samples arriving each day.

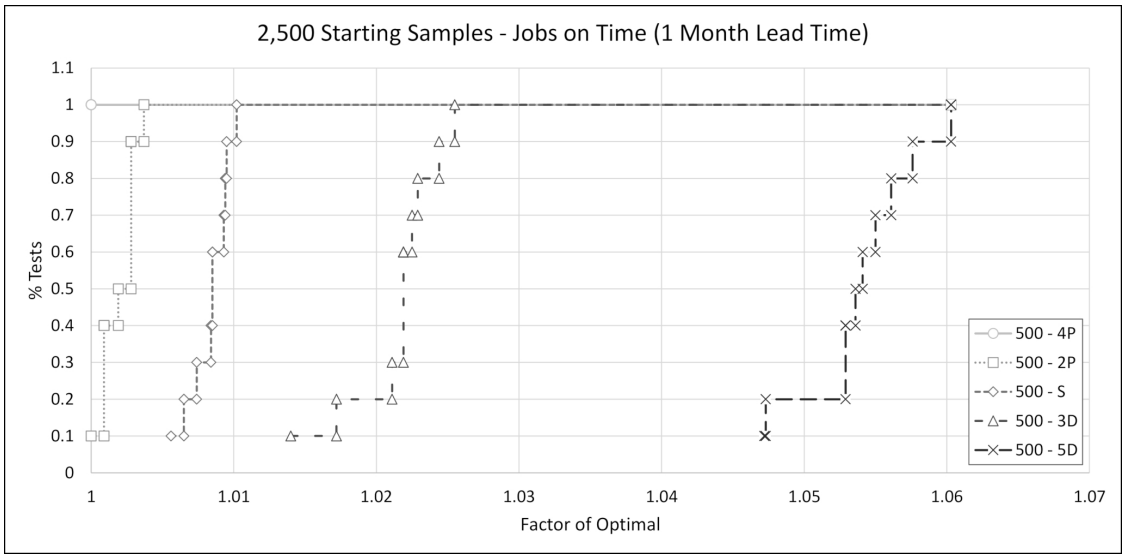


Figure B.6: Proportion of jobs on time for one month lead times and instances starting with 2,500 samples and 500 samples arriving each day.

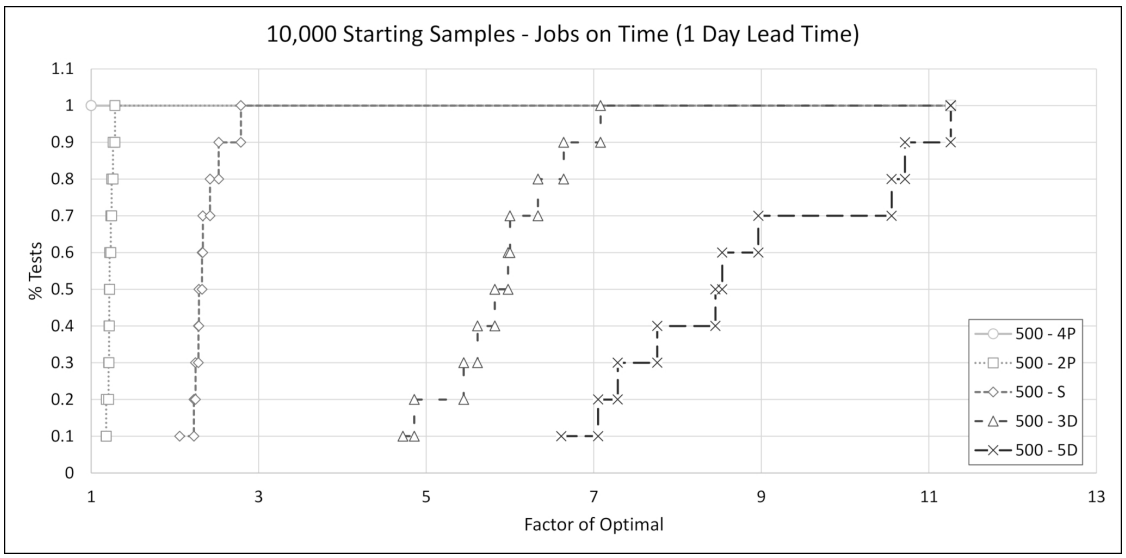


Figure B.7: Proportion of jobs on time for one day lead times and instances starting with 10,000 samples and 500 samples arriving each day.

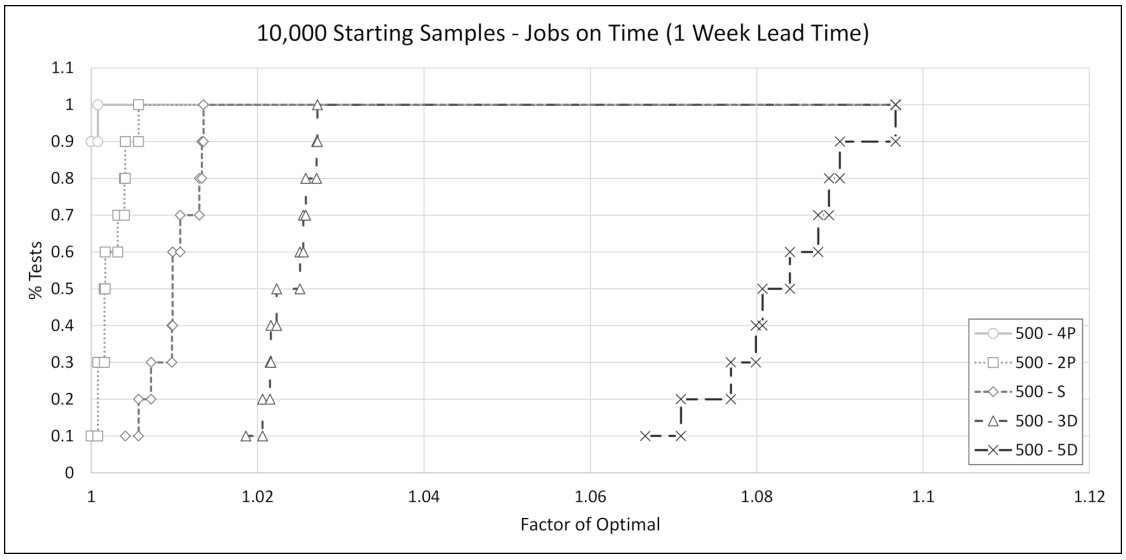


Figure B.8: Proportion of jobs on time for one week lead times and instances starting with 10,000 samples and 500 samples arriving each day.

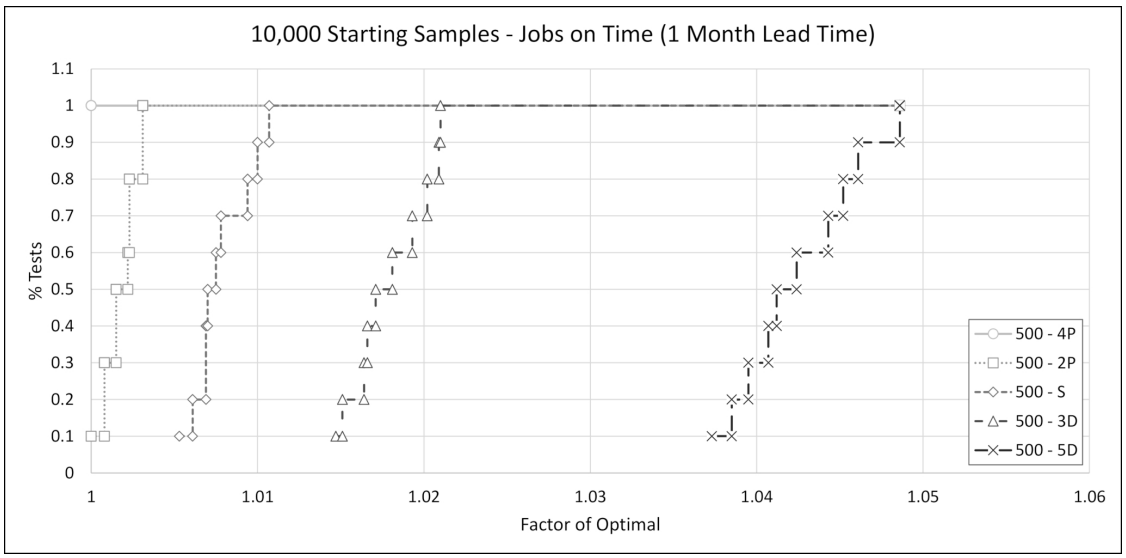


Figure B.9: Proportion of jobs on time for one month lead times and instances starting with 10,000 samples and 500 samples arriving each day.

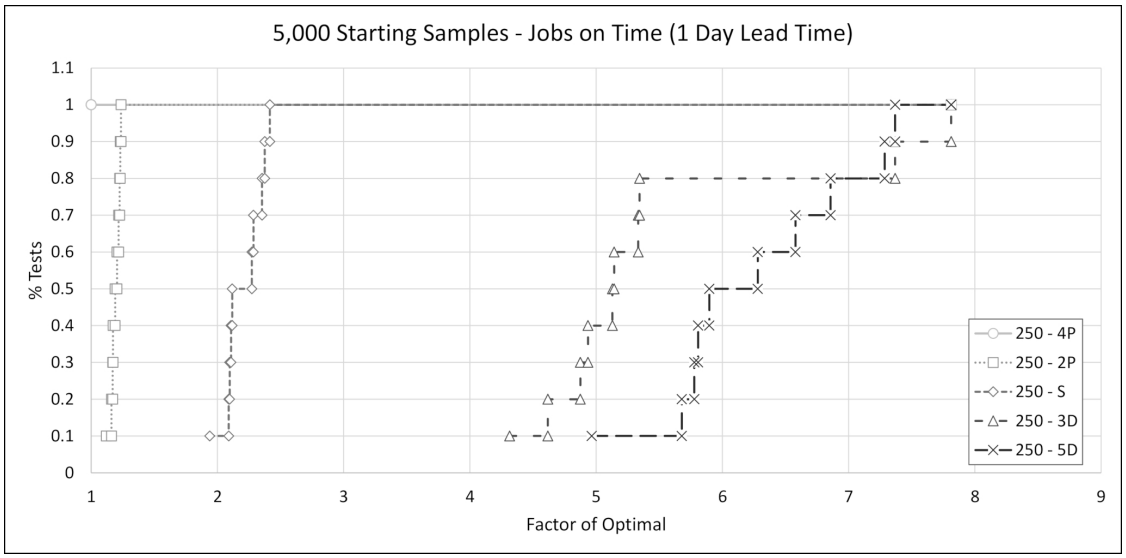


Figure B.10: Proportion of jobs on time for one day lead times and instances starting with 5,000 samples and 250 samples arriving each day.

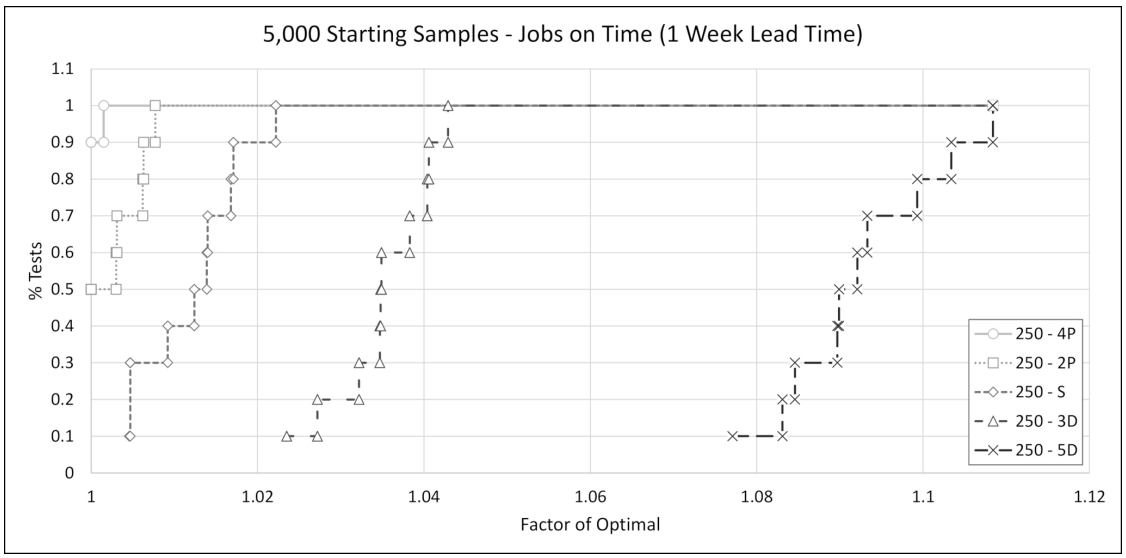


Figure B.11: Proportion of jobs on time for one week lead times and instances starting with 5,000 samples and 250 samples arriving each day.

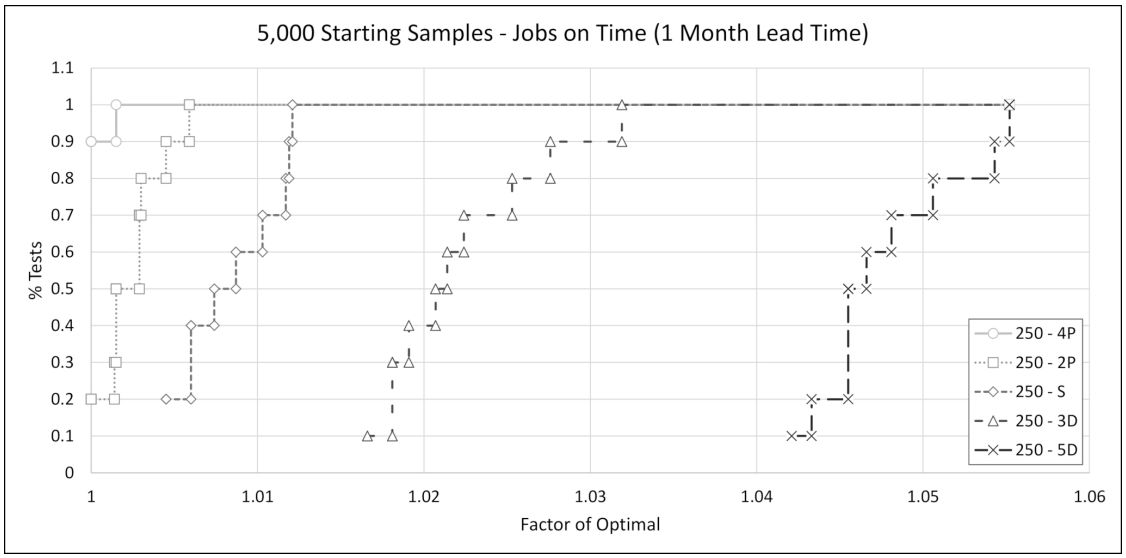


Figure B.12: Proportion of jobs on time for one month lead times and instances starting with 5,000 samples and 250 samples arriving each day.

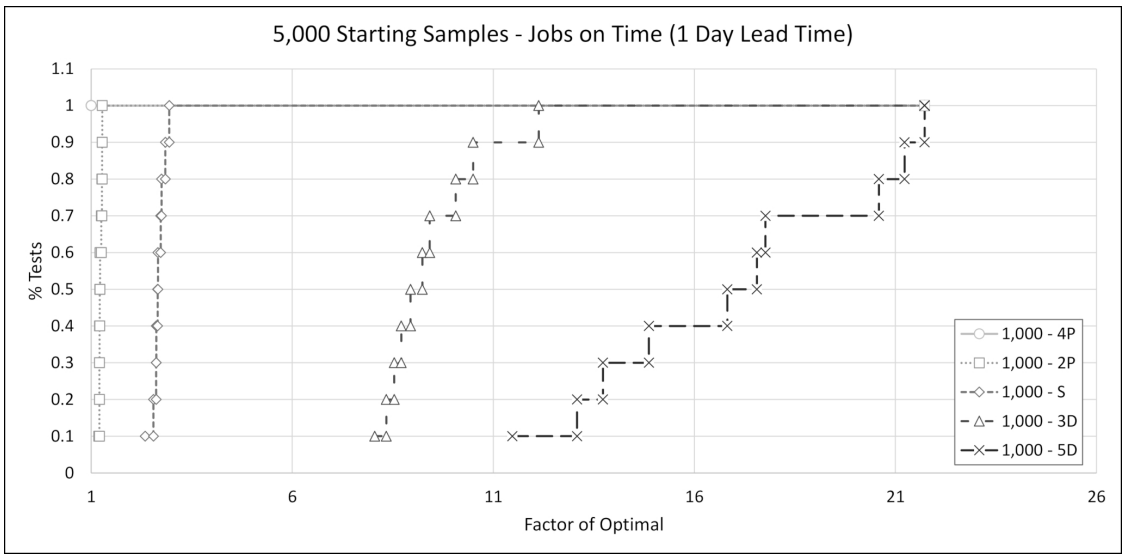


Figure B.13: Proportion of jobs on time for one day lead times and instances starting with 5,000 samples and 1,000 samples arriving each day.

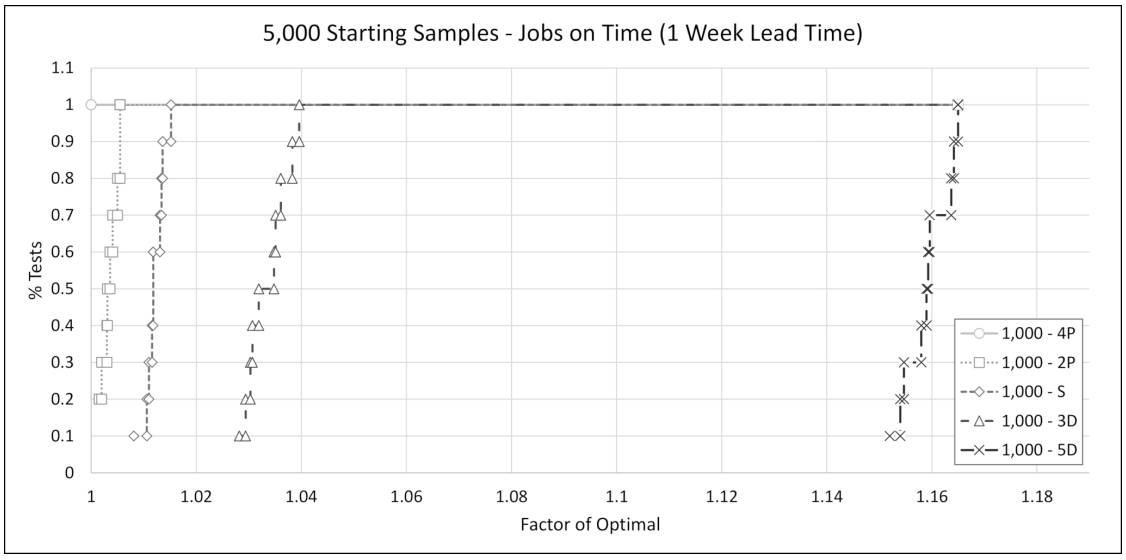


Figure B.14: Proportion of jobs on time for one week lead times and instances starting with 5,000 samples and 1,000 samples arriving each day.

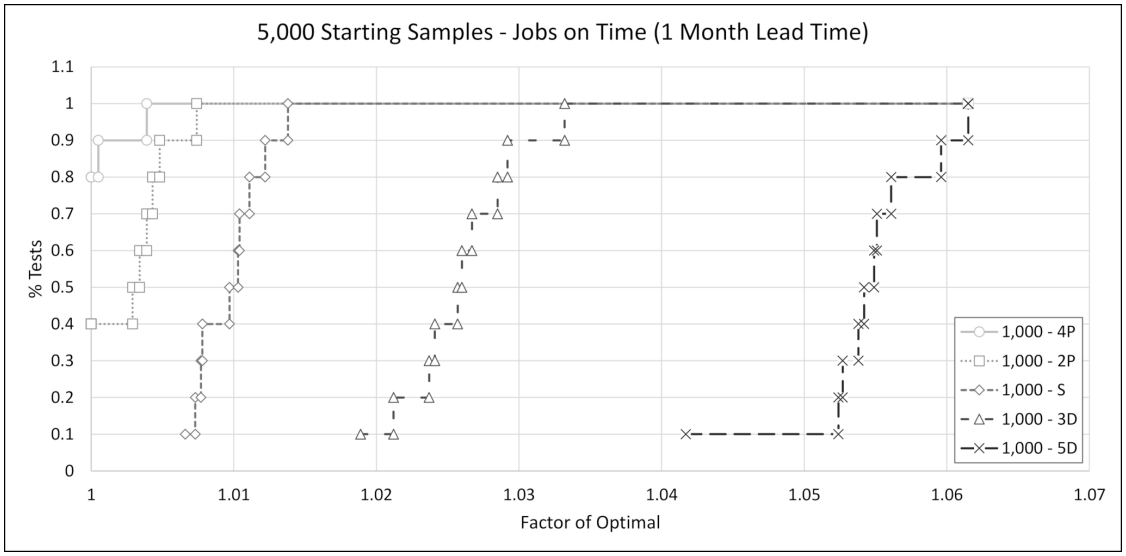


Figure B.15: Proportion of jobs on time for one month lead times and instances starting with 5,000 samples and 1,000 samples arriving each day.

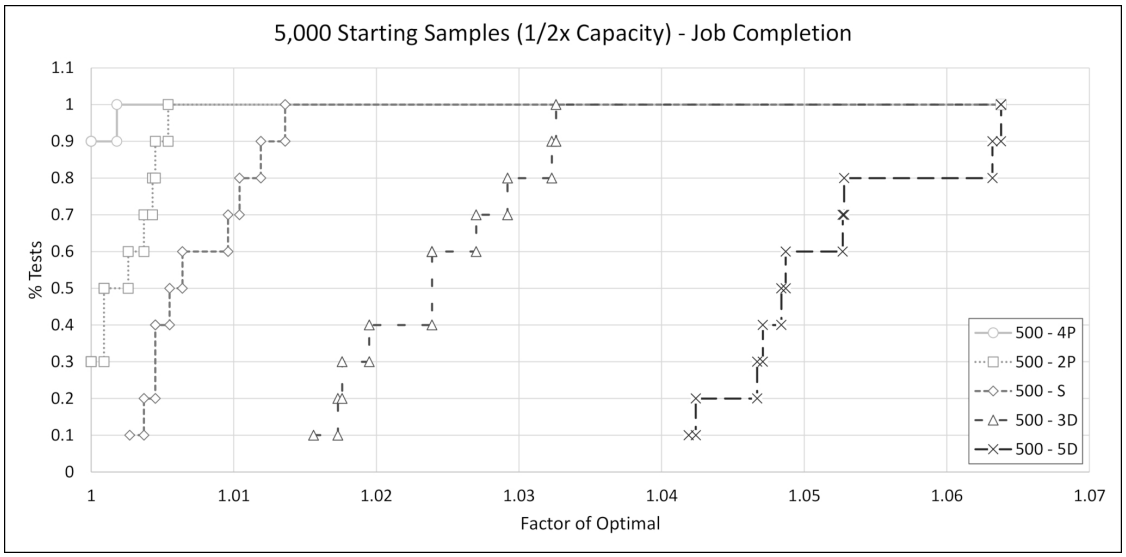


Figure B.16: Performance profile comparing job completion among rescheduling policies for instances with one half the original capacity starting with 5,000 samples and 500 samples arriving each day.

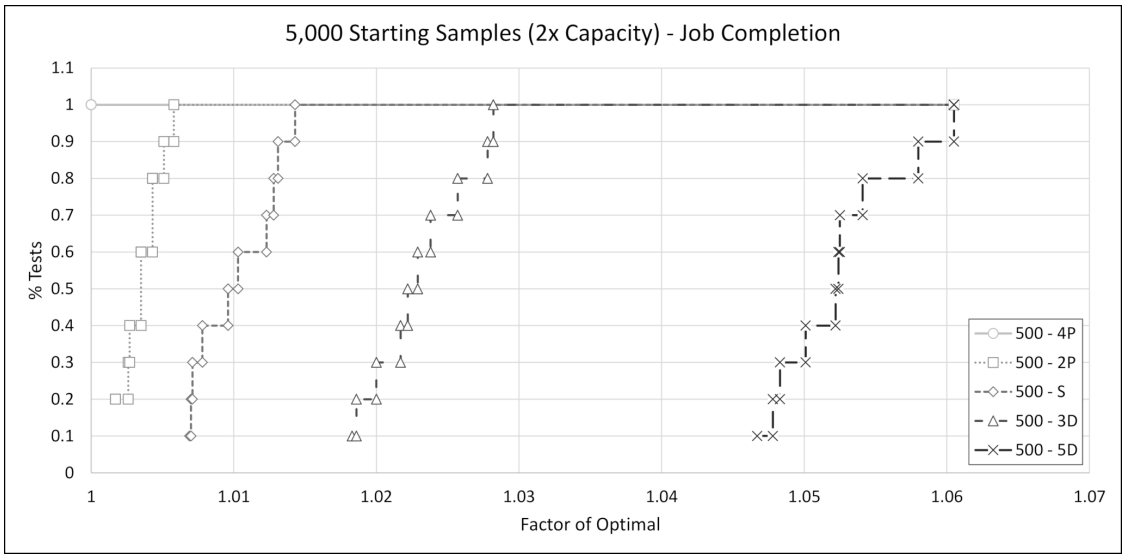


Figure B.17: Performance profile comparing job completion among rescheduling policies for instances with twice the original capacity starting with 5,000 samples and 500 samples arriving each day.

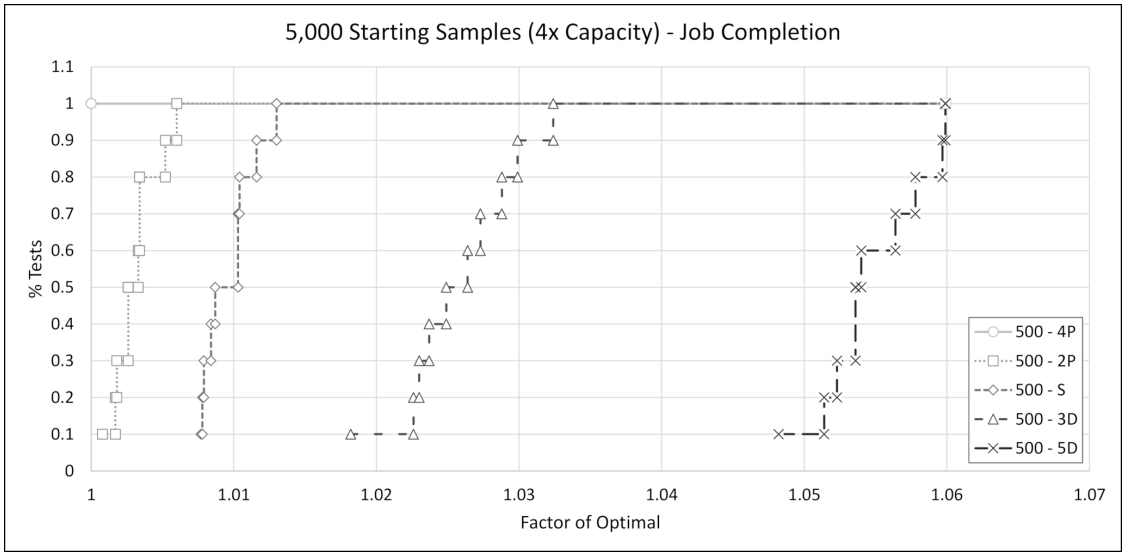


Figure B.18: Performance profile comparing job completion among rescheduling policies for instances with four times the original capacity starting with 5,000 samples and 500 samples arriving each day.

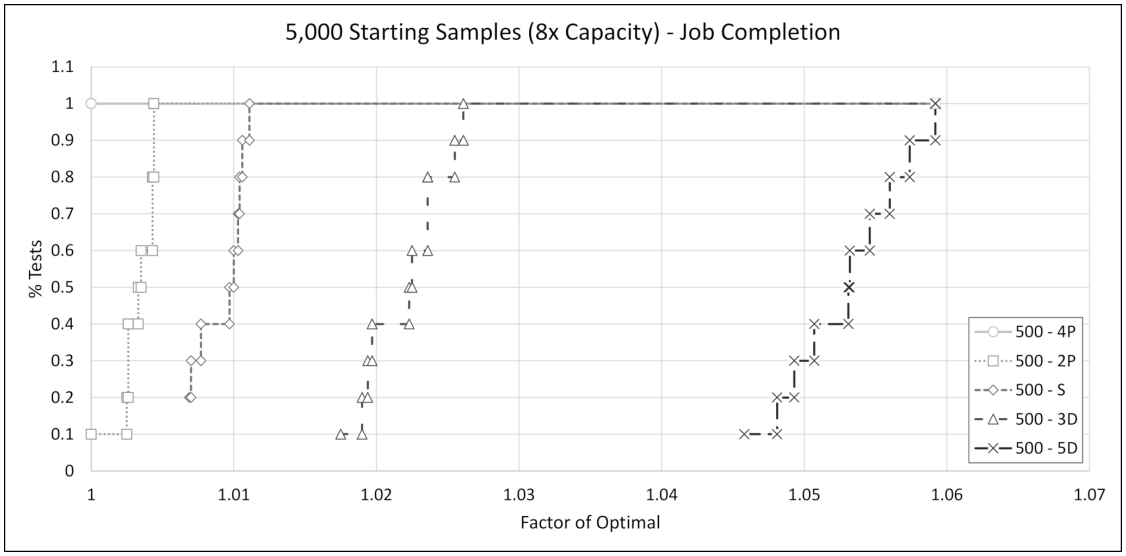


Figure B.19: Performance profile comparing job completion among rescheduling policies for instances with eight times the original capacity starting with 5,000 samples and 500 samples arriving each day.

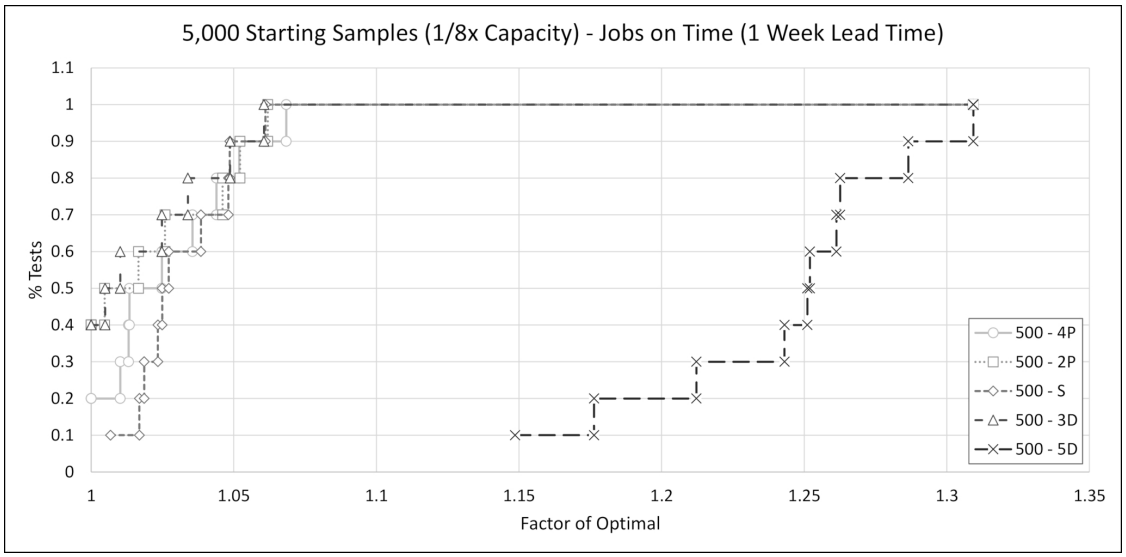


Figure B.20: Proportion of jobs on time for one week lead times and instances with one eighth the original capacity starting with 5,000 samples and 500 samples arriving each day.

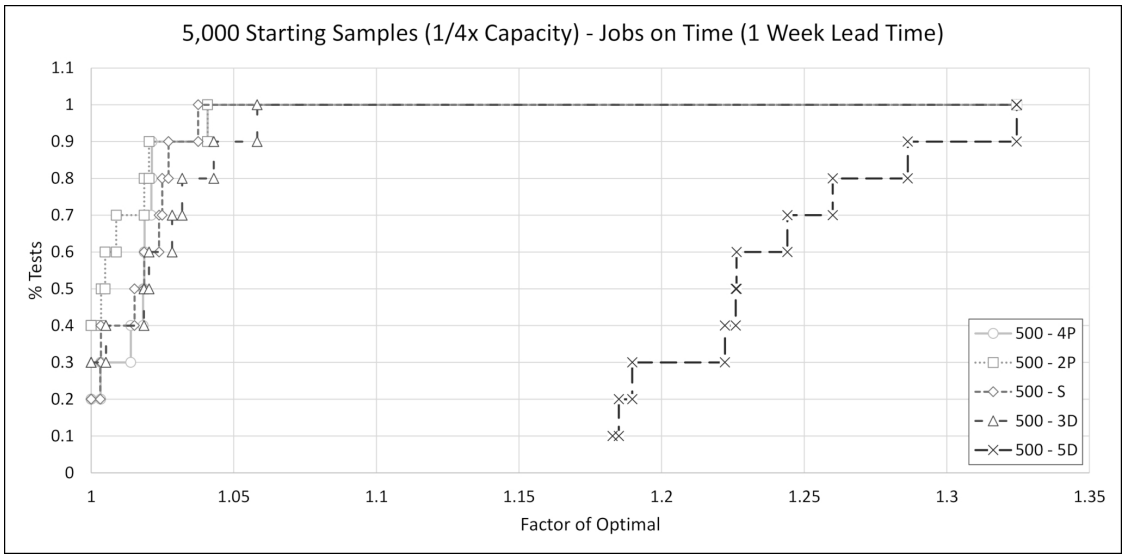


Figure B.21: Proportion of jobs on time for one week lead times and instances with one quarter the original capacity starting with 5,000 samples and 500 samples arriving each day.

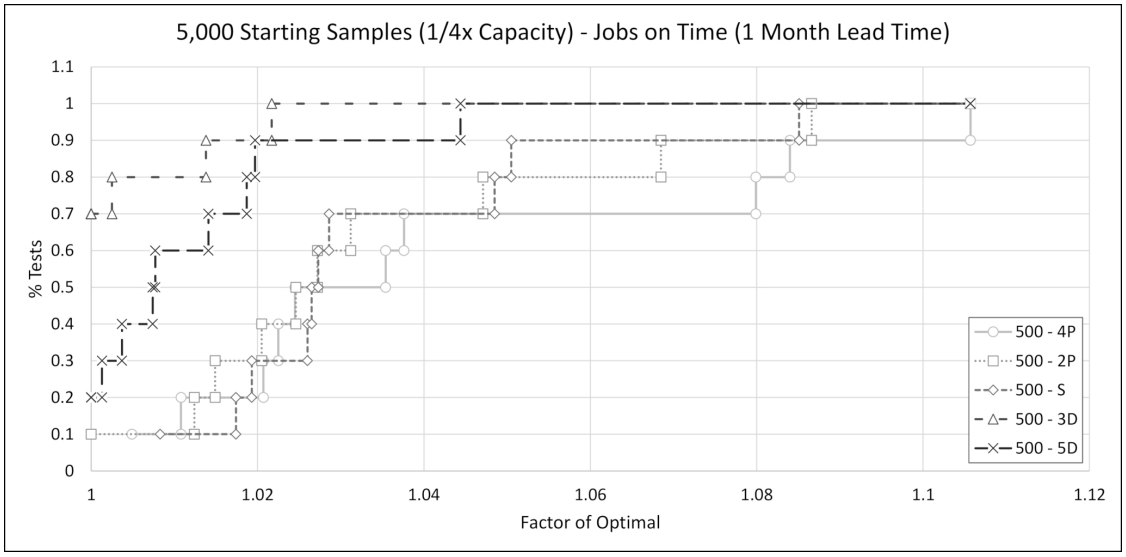


Figure B.22: Proportion of jobs on time for one month lead times and instances with one quarter the original capacity starting with 5,000 samples and 500 samples arriving each day.

Appendix C

Dynamic Timepoint Algorithms

Algorithm Dynamic Timepoint Framework shown in Table C.1 describes the framework discussed in Section 4.2 in more detail. Algorithm Update Solution shown in Table C.2 demonstrates how we update the solution from our current iteration to be feasible for the proceeding problem. The functions included in algorithm Helper Methods shown in Table C.3 are used by the main framework in algorithm Dynamic Timepoint Framework.

Table C.1: Algorithm 5, Dynamic Timepoint Framework

Algorithm 5 Dynamic Timepoint Framework

procedure ITERATIVELY SOLVE(G, ε, x_0, `its_tl`, `obj_thresh`, `sols_tl`, `fin_tl`, `fin_disc`)
▷ Description of parameters given in Table 4.1
    `start_time` ← NOW( )
    $x \leftarrow x_0$ ▷ x_0 is an initial solution, if provided
    `stop_iterating` ← false
    while `stop_iterating` = false **do**
        $G^* \leftarrow$ BUILD GRAPH(G, ε) ▷ Build G^*
        $X \leftarrow$ SOLVE PROBLEM(G^*, f_{G^*}, C_{G^*}, x, `its_tl`, `sols_tl`) ▷ Solve ($P2$)
        if $X \neq \emptyset$ **then** ▷ A solution was found
            $\varepsilon^+ \leftarrow \{\varepsilon^+(u) = \emptyset : u \in V\}$; $\varepsilon^- \leftarrow \{\varepsilon^-(u) = \{S, S+1, S+2, \ldots, S+H\} : u \in V\}$ ▷ Initialize timepoint modification sets to be empty
            for all $x^* \in X$ **do** ▷ For all found solutions
                $(\varepsilon^+_{x^*}, \varepsilon^-_{x^*}) \leftarrow$ GET DYNAMIC TIMEPOINTS(G, G^*, ε, x^*) ▷ Get timepoint modifications
                $\varepsilon^+ \leftarrow \{\varepsilon^+(u) \cup \varepsilon^+_{x^*}(u) : u \in V\}$; $\varepsilon^- \leftarrow \{\varepsilon^-(u) \cap \varepsilon^-_{x^*}(u) : u \in V\}$ ▷ Collate timepoint modifications
            end for
            $\varepsilon^* \leftarrow \{\varepsilon(u) \cup \varepsilon^+(u) \setminus \varepsilon^-(u) : u \in V\}$ ▷ Construct next set of timepoints
            $G^{**} \leftarrow$ BUILD GRAPH(G, ε^*) ▷ Build graph for next iteration
            $x^* \leftarrow$ GET MODIFIED SOLUTION($x, G^*, G^{**}, \varepsilon, \varepsilon^+, \varepsilon^-$) ▷ Modify best solution from current iteration to be feasible for next iteration
            `stop_iterating` ← CHECK STOPPING CRITERIA($x, x^*, f_{G^*}, f_{G^{**}}$, `start_time`, `obj_thresh`, `its_tl`)
            $x \leftarrow x^*$; $\varepsilon \leftarrow \varepsilon^*$
        else ▷ We did not find any solutions
            `stop_iterating` ← true
        end if
    end while
    $\varepsilon^+ \leftarrow$ `fin_disc`
    $\varepsilon^* \leftarrow \{\varepsilon(u) \cup \varepsilon^+(u) : u \in V\}$ ▷ Add timepoints from discretization assumed to give acceptable solutions
    $G^{**} \leftarrow$ BUILD GRAPH(G, ε^*)
    $x^* \leftarrow$ GET MODIFIED SOLUTION($x, G^*, G^{**}, \varepsilon, \varepsilon^+, \varepsilon^-$)
    $X \leftarrow$ SOLVE PROBLEM(G^*, f_{G^*}, x, `fin_tl`, ∞) ▷ Solve problem with many timepoints
    return X
end procedure

Table C.2: Algorithm 6, Update Solution

Algorithm 6 Update Solution

function GET MODIFIED SOLUTION($x, G^*, G^{**}, \varepsilon, \varepsilon^+, \varepsilon^-$)
  $x^* \leftarrow 0$ ▷ Initialize new solution to be of correct dimension, with all entries 0
  SET ACTIVE ARCS($x, G^*, \varepsilon, \varepsilon^+, \varepsilon^-, x^*$) ▷ Set the x variable entries
  SET ACTIVE RESOURCES($x, G^*, \varepsilon, \varepsilon^+, \varepsilon^-, x^*$) ▷ Set the y variable entries
  SET INACTIVE ARCS($x, G^*, \varepsilon, \varepsilon^+, \varepsilon^-, x^*$) ▷ Set the w variable entries
  return x^*
end function
procedure SET ACTIVE ARCS($x, G^*, \varepsilon, \varepsilon^+, \varepsilon^-, x^*$)
  for all $i \in I$ **do**
    for all $u \in \Pi(i)$ **do**
      for all $t \in \varepsilon(u)$ **do**
        $x^*(i, u, t) \leftarrow x(i, u, t)$ ▷ Set all active arcs from previous solution
      end for
    end for
  end for
end procedure
procedure SET ACTIVE RESOURCES($x, G^*, \varepsilon, \varepsilon^+, \varepsilon^-, x^*$)
  for all $(u, t) \in V^*$ **do**
    for all $v \in N_G^+(u)$ **do**
      $y^*(u, v, t) \leftarrow y(u, v, t)$ ▷ Set all resource usage values from previous solution. Any new timepoints are already initialized to have 0 resource usage
    end for
  end for
end procedure
procedure SET INACTIVE ARCS($x, G^*, \varepsilon, \varepsilon^+, \varepsilon^-, x^*$)
  for all $i \in I$ **do**
    for all $k \in \{1, \ldots, |\Pi(i)|\}$ **do**
      $u \leftarrow \Pi(i, k)$
      for all $k \in \{1, \ldots, |\varepsilon(u)|\}$ **do**
        $t \leftarrow \varepsilon(u, k)$; $t^- \leftarrow \varepsilon(u, k-1)$
        $w^*(i, u, t) \leftarrow w^*(i, u, t^-) - x^*(i, u, t) + \sum x^*(i, v, t') : H(i, v, t') = (u, t)$ ▷ Add material that waited previously or arrived from earlier state, and subtract material leaving state
      end for
    end for
  end for
end procedure

Table C.3: Algorithm 7, Helper Methods

Algorithm 7 Helper Methods

function GET DYNAMIC TIMEPOINTS(G, G^*, ε, x)
    $\varepsilon^+, \varepsilon^- \leftarrow \{\emptyset : u \in V\}$ ▷ Initialize timepoint modifications sets to be empty
    $\varepsilon_1^+ \leftarrow \cup$ GET INSTANT START TIMEPOINTS(G, G^*, ε, x)
    $\varepsilon_2^+ \leftarrow \cup$ GET OVERLOADED TIMEPOINTS(G, G^*, ε, x)
    $\varepsilon_1^- \leftarrow \cup$ GET DOMINATED TIMEPOINTS(G, G^*, ε, x)
    $\varepsilon^+ \leftarrow \{\varepsilon^+(u) \cup \varepsilon_1^+(u) \cup \varepsilon_2^+(u) : u \in V\}$ ▷ Collate timepoints to add
    $\varepsilon^- \leftarrow \{\varepsilon^-(u) \cup \varepsilon_1^-(u) : u \in V\}$ ▷ Collate timepoints to remove
    return $(\varepsilon^+, \varepsilon^-)$
end function
function CHECK STOPPING CRITERIA($x, x^*, f_{G^*}, f_{G^{**}}$, `start_time`, `obj_thresh`, `its_tl`)
    if $(f_{G^{**}}(x^*) - f_{G^*}(x))/f_{G^*}(x) <$ `obj_thresh` **then** ▷ Insufficient progress made
        return true
    end if
    if NOW( )$-$`start_time` $>$ `its_tl` **then** ▷ Iterations time limit exceeded
        return true
    end if
    return false
end function
function SOLVE PROBLEM(G^*, f_{G^*}, x, `fin_tl`, `sols_tl`)
    $P2 \leftarrow \max f_{G^*}(x)$: all constraints are satisfied ▷ Define problem $P2$
    $X \leftarrow$ SOLVE(P2, x, `fin_tl`, `sols_tl`) ▷ Use a solver to solve $P2$, provide x as initial solution, use time limit `fin_tl`, return if elapsed time since last found solution surpasses `sols_tl`
    return X ▷ X is set of found solutions (possibly empty)
end function

Appendix D

Dynamic Timepoint Instances

Table D.1 presents detailed information about the results that were reported in Section 4.3 concerning the performance of dynamic timepoint policies. The first column is the instance number of the entry, the second column refers to the amount of elapsed time the entry corresponds to, and the third and fourth columns include the size of the instance when using the static NUD60 discretization. The fifth and sixth columns refer to the number of days that were scheduled for the instances and the number of samples that were scheduled. The seventh column displays the size category that the instance was sorted into. The remaining columns indicate the relative performance of each policy at the time of the relevant checkpoint.

Table D.1: Results for each instance of dynamic timepoint study.

Instance	Time Elapsed (Min.)	# Vars.	# Cons.	Length of Horizon (Days)	# Samples	Size	NUD60	UD60	UD120	UD240	5 - 0 - UD60	5 - 0 - UD120	5 - 0 - UD240	60 - 1.05 - UD240
1	1	698,866	388,218	1	10,000	Small	0.954	0.954	0.954	0.851	0.851	0.851	0.682	0.682
1	5	698,866	388,218	1	10,000	Small	0.954	0.954	0.851	0.851	0.682	0.682	0.486	0.486
1	15	698,866	388,218	1	10,000	Small	0.954	0.851	0.851	0.682	0.486	0.486	0.975	0.948
1	30	698,866	388,218	1	10,000	Small	0.954	0.851	0.682	0.486	0	0.975	0.984	1

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Time Elapsed (Min.)	# Vars.	# Cons.	Length of Horizon (Days)	# Samples	Size	NUD60	UD60	UD120	UD240	5 - 0 - UD60	5 - 0 - UD120	5 - 0 - UD240	60 - 1.05 - UD240
1	60	698,866	388,218	1	10,000	Small	0.954	0.851	0.682	0.486	0.975	0.984	1	0.973
2	1	686,782	382,158	1	10,000	Small	0.933	0.933	0.933	0.841	0.841	0.841	0.667	0.667
2	5	686,782	382,158	1	10,000	Small	0.933	0.933	0.841	0.841	0.667	0.667	0.474	0.474
2	15	686,782	382,158	1	10,000	Small	0.933	0.841	0.841	0.667	0.474	0.474	1	0.922
2	30	686,782	382,158	1	10,000	Small	0.933	0.841	0.667	0.474	0	1	0.991	0.993
2	60	686,782	382,158	1	10,000	Small	0.933	0.841	0.667	0.474	1	0.991	0.993	0.98
3	1	657,822	367,693	1	10,000	Small	0.953	0.953	0.953	0.842	0.842	0.842	0.658	0.658
3	5	657,822	367,693	1	10,000	Small	0.953	0.953	0.842	0.842	0.658	0.658	0.461	0.461
3	15	657,822	367,693	1	10,000	Small	0.953	0.842	0.842	0.658	0.461	0.461	0.98	0.924
3	30	657,822	367,693	1	10,000	Small	0.953	0.842	0.658	0.461	0	0.98	0.976	1
3	60	657,822	367,693	1	10,000	Small	0.953	0.842	0.658	0.461	0.98	0.976	1	0.998
4	1	646,562	361,874	1	10,000	Small	0.973	0.973	0.973	0.864	0.864	0.864	0.684	0.684
4	5	646,562	361,874	1	10,000	Small	0.973	0.973	0.864	0.864	0.684	0.684	0.487	0.487
4	15	646,562	361,874	1	10,000	Small	0.973	0.864	0.864	0.684	0.487	0.487	0.985	0.939
4	30	646,562	361,874	1	10,000	Small	0.973	0.864	0.684	0.487	0	0.985	0.984	0.999
4	60	646,562	361,874	1	10,000	Small	0.973	0.864	0.684	0.487	0.985	0.984	0.999	1
5	1	662,390	369,871	1	10,000	Small	0.956	0.956	0.956	0.839	0.839	0.839	0.669	0.669
5	5	662,390	369,871	1	10,000	Small	0.956	0.956	0.839	0.839	0.669	0.669	0.469	0.469
5	15	662,390	369,871	1	10,000	Small	0.956	0.839	0.839	0.669	0.469	0.469	0.974	0.946
5	30	662,390	369,871	1	10,000	Small	0.956	0.839	0.669	0.469	0	0.974	0.987	1
5	60	662,390	369,871	1	10,000	Small	0.956	0.839	0.669	0.469	0.974	0.987	1	0.975
6	1	1,245,832	664,038	1	20,000	Med.	0.955	0.955	0.955	0.832	0.832	0.832	0.68	0.68
6	5	1,245,832	664,038	1	20,000	Med.	0.955	0.955	0.832	0.832	0.68	0.68	0.525	0.525
6	15	1,245,832	664,038	1	20,000	Med.	0.955	0.832	0.832	0.68	0.525	0.525	0.967	0.807
6	30	1,245,832	664,038	1	20,000	Med.	0.955	0.832	0.68	0.525	1	0.967	0.994	1
6	60	1,245,832	664,038	1	20,000	Med.	0.955	0.832	0.68	0.525	0.967	0.994	1	0.998
7	1	1,318,834	700,531	1	20,000	Med.	0.932	0.932	0.932	0.808	0.808	0.808	0.651	0.651
7	5	1,318,834	700,531	1	20,000	Med.	0.932	0.932	0.808	0.808	0.651	0.651	0.479	0.479
7	15	1,318,834	700,531	1	20,000	Med.	0.932	0.808	0.808	0.651	0.479	0.479	0.991	0.877
7	30	1,318,834	700,531	1	20,000	Med.	0.932	0.808	0.651	0.479	1	0.991	0.982	1
7	60	1,318,834	700,531	1	20,000	Med.	0.932	0.808	0.651	0.479	0.991	0.982	1	0.987
8	1	1,235,072	658,557	1	20,000	Med.	0.928	0.936	0.936	0.827	0.827	0.827	0.67	0.67

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Time Elapsed (Min.)	# Vars.	# Cons.	Length of Horizon (Days)	# Samples	Size	NUD60	UD60	UD120	UD240	5 - 0 - UD60	5 - 0 - UD120	5 - 0 - UD240	60 - 1.05 - UD240
8	5	1,235,072	658,557	1	20,000	Med.	0.936	0.936	0.827	0.827	0.67	0.67	0.497	0.497
8	15	1,235,072	658,557	1	20,000	Med.	0.936	0.827	0.827	0.67	0.497	0.497	0.976	0.905
8	30	1,235,072	658,557	1	20,000	Med.	0.936	0.827	0.67	0.497	1	0.976	0.985	1
8	60	1,235,072	658,557	1	20,000	Med.	0.936	0.827	0.67	0.497	0.976	0.985	1	0.976
9	1	1,146,524	614,159	1	20,000	Med.	0.951	0.951	0.951	0.828	0.828	0.828	0.678	0.678
9	5	1,146,524	614,159	1	20,000	Med.	0.951	0.951	0.828	0.828	0.678	0.678	0.514	0.514
9	15	1,146,524	614,159	1	20,000	Med.	0.951	0.828	0.828	0.678	0.514	0.514	0.989	0.922
9	30	1,146,524	614,159	1	20,000	Med.	0.951	0.828	0.678	0.514	1	0.989	0.983	1
9	60	1,146,524	614,159	1	20,000	Med.	0.951	0.828	0.678	0.514	0.989	0.983	1	0.975
10	1	1,337,332	709,802	1	20,000	Med.	0.938	0.938	0.938	0.82	0.82	0.82	0.667	0.667
10	5	1,337,332	709,802	1	20,000	Med.	0.938	0.938	0.82	0.82	0.667	0.667	0.499	0.499
10	15	1,337,332	709,802	1	20,000	Med.	0.938	0.82	0.82	0.667	0.499	0.499	0.948	0.904
10	30	1,337,332	709,802	1	20,000	Med.	0.938	0.82	0.667	0.499	1	0.948	0.987	1
10	60	1,337,332	709,802	1	20,000	Med.	0.938	0.82	0.667	0.499	0.948	0.987	1	0.993
11	1	1,973,660	1,030,570	1	30,000	Med.	0.941	0.941	0.941	0.78	0.78	0.78	0.649	0.649
11	5	1,973,660	1,030,570	1	30,000	Med.	0.941	0.941	0.78	0.78	0.649	0.649	0.5	0.5
11	15	1,973,660	1,030,570	1	30,000	Med.	0.941	0.78	0.78	0.649	0.5	0.5	0.985	0.761
11	30	1,973,660	1,030,570	1	30,000	Med.	0.941	0.78	0.649	0.5	1	0.985	0.97	0.998
11	60	1,973,660	1,030,570	1	30,000	Med.	0.941	0.78	0.649	0.5	0.985	0.97	1	0.975
12	1	1,897,032	992,192	1	30,000	Med.	0.934	0.934	0.934	0.799	0.799	0.799	0.661	0.661
12	5	1,897,032	992,192	1	30,000	Med.	0.934	0.934	0.799	0.799	0.661	0.661	0.509	0.509
12	15	1,897,032	992,192	1	30,000	Med.	0.934	0.799	0.799	0.661	0.509	0.509	0.975	0.767
12	30	1,897,032	992,192	1	30,000	Med.	0.934	0.799	0.661	0.509	2	0.975	0.964	1
12	60	1,897,032	992,192	1	30,000	Med.	0.934	0.799	0.661	0.509	0.975	0.964	1	0.985
13	1	1,856,322	971,578	1	30,000	Med.	0.932	0.932	0.932	0.791	0.791	0.791	0.667	0.667
13	5	1,856,322	971,578	1	30,000	Med.	0.932	0.932	0.791	0.791	0.667	0.667	0.516	0.516
13	15	1,856,322	971,578	1	30,000	Med.	0.932	0.791	0.791	0.667	0.516	0.516	0.938	0.667
13	30	1,856,322	971,578	1	30,000	Med.	0.932	0.791	0.667	0.516	1	0.938	0.962	1
13	60	1,856,322	971,578	1	30,000	Med.	0.932	0.791	0.667	0.516	0.938	0.962	1	0.966
14	1	1,887,126	987,090	1	30,000	Med.	0.936	0.936	0.936	0.782	0.782	0.782	0.651	0.651
14	5	1,887,126	987,090	1	30,000	Med.	0.936	0.936	0.782	0.782	0.651	0.651	0.501	0.501
14	15	1,887,126	987,090	1	30,000	Med.	0.936	0.782	0.782	0.651	0.501	0.501	0.967	0.881

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Time Elapsed (Min.)	# Vars.	# Cons.	Length of Horizon (Days)	# Samples	Size	NUD60	UD60	UD120	UD240	5 - 0 - UD60	5 - 0 - UD120	5 - 0 - UD240	60 - 1.05 - UD240
14	30	1,887,126	987,090	1	30,000	Med.	0.936	0.782	0.651	0.501	1	0.967	0.97	1
14	60	1,887,126	987,090	1	30,000	Med.	0.936	0.782	0.651	0.501	0.967	0.97	1	0.969
15	1	1,931,448	1,009,384	1	30,000	Med.	0.921	0.921	0.921	0.785	0.785	0.785	0.654	0.654
15	5	1,931,448	1,009,384	1	30,000	Med.	0.921	0.921	0.785	0.785	0.654	0.654	0.503	0.503
15	15	1,931,448	1,009,384	1	30,000	Med.	0.921	0.785	0.785	0.654	0.503	0.503	0.953	0.865
15	30	1,931,448	1,009,384	1	30,000	Med.	0.921	0.785	0.654	0.503	1	0.953	0.96	0.998
15	60	1,931,448	1,009,384	1	30,000	Med.	0.921	0.785	0.654	0.503	0.953	0.96	1	0.964
16	1	1,267,144	707,713	2	10,000	Med.	0	1	1	0	0.868	0.868	0.743	0.743
16	5	1,267,144	707,713	2	10,000	Med.	0.987	1	0.868	0.868	0.743	0.743	0.515	0.515
16	15	1,267,144	707,713	2	10,000	Med.	1	0	0.868	0.743	0.515	0.515	0.989	0.743
16	30	1,267,144	707,713	2	10,000	Med.	1	0.868	0.743	0.515	2	0.989	0.977	0.96
16	60	1,267,144	707,713	2	10,000	Med.	1	0.868	0.743	0.515	0.989	0.977	0.96	0.963
17	1	1,268,814	708,545	2	10,000	Med.	0	1	1	0	0.866	0.866	0.733	0.733
17	5	1,268,814	708,545	2	10,000	Med.	0.989	1	0.866	0.866	0.733	0.733	0.515	0.515
17	15	1,268,814	708,545	2	10,000	Med.	1	0	0.866	0.733	0.515	0.515	0.982	0.764
17	30	1,268,814	708,545	2	10,000	Med.	1	0.866	0.733	0.515	2	0.982	0.977	0.981
17	60	1,268,814	708,545	2	10,000	Med.	1	0.866	0.733	0.515	0.982	0.977	0.981	0.951
18	1	1,347,948	748,069	2	10,000	Med.	0	0.992	0.992	0	0.828	0.828	0.718	0.718
18	5	1,347,948	748,069	2	10,000	Med.	0.979	0.992	0.828	0.828	0.718	0.718	0.508	0.508
18	15	1,347,948	748,069	2	10,000	Med.	0.992	0	0.828	0.718	0.508	0.508	0.985	0.95
18	30	1,347,948	748,069	2	10,000	Med.	0.992	0.828	0.718	0.508	2	1	0.973	0.965
18	60	1,347,948	748,069	2	10,000	Med.	0.992	0.828	0.718	0.508	1	0.973	0.965	0.916
19	1	1,360,774	754,589	2	10,000	Med.	0	1	1	0.845	0.854	0.854	0.714	0.714
19	5	1,360,774	754,589	2	10,000	Med.	0.989	1	0.854	0.854	0.714	0.714	0.516	0.516
19	15	1,360,774	754,589	2	10,000	Med.	1	0.845	0.854	0.714	0.516	0.516	0.988	0.898
19	30	1,360,774	754,589	2	10,000	Med.	1	0.854	0.714	0.516	2	0.988	0.932	0.967
19	60	1,360,774	754,589	2	10,000	Med.	1	0.854	0.714	0.516	0.988	0.932	0.967	0.948
20	1	1,259,862	704,040	2	10,000	Med.	0	0.993	0.993	0	0.869	0.869	0.764	0.764
20	5	1,259,862	704,040	2	10,000	Med.	0.993	0.993	0.869	0.869	0.764	0.764	0.51	0.51
20	15	1,259,862	704,040	2	10,000	Med.	0.993	0	0.869	0.764	0.51	0.51	0.987	0.764
20	30	1,259,862	704,040	2	10,000	Med.	0.993	0.869	0.764	0.51	3	0.987	0.961	0.961
20	60	1,259,862	704,040	2	10,000	Med.	0.993	0.869	0.764	0.51	0.987	0.961	1	0.961

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Time Elapsed (Min.)	# Vars.	# Cons.	Length of Horizon (Days)	# Samples	Size	NUD60	UD60	UD120	UD240	5 - 0 - UD60	5 - 0 - UD120	5 - 0 - UD240	60 - 1.05 - UD240
21	1	1,910,338	1,030,553	2	15,000	Med.	0	1	1	0	0.863	0.863	0	0.719
21	5	1,910,338	1,030,553	2	15,000	Med.	0	1	0	0.863	0.719	0.719	0.496	0.496
21	15	1,910,338	1,030,553	2	15,000	Med.	1	0	0.863	0.719	0.496	0.496	0.995	0.719
21	30	1,910,338	1,030,553	2	15,000	Med.	1	0.863	0.719	0.496	11	0.995	0.954	0.932
21	60	1,910,338	1,030,553	2	15,000	Med.	1	0.863	0.719	0.496	0.995	0.954	0.932	0.937
22	1	1,834,310	992,558	2	15,000	Med.	0	1	1	0	0.9	0.9	0.726	0.726
22	5	1,834,310	992,558	2	15,000	Med.	0	1	0.898	0.9	0.726	0.726	0.511	0.511
22	15	1,834,310	992,558	2	15,000	Med.	1	0	0.9	0.726	0.511	0.511	0.992	0.726
22	30	1,834,310	992,558	2	15,000	Med.	1	0.9	0.726	0.511	9	0.992	0.96	0.93
22	60	1,834,310	992,558	2	15,000	Med.	1	0.9	0.726	0.511	0.992	0.96	0.98	0.897
23	1	1,949,002	1,049,765	2	15,000	Med.	0	0.996	0.996	0	0.831	0.831	0.685	0.685
23	5	1,949,002	1,049,765	2	15,000	Med.	0	0.996	0	0.831	0.685	0.685	0.48	0.48
23	15	1,949,002	1,049,765	2	15,000	Med.	0.996	0	0.831	0.685	0.48	0.48	0.986	0.685
23	30	1,949,002	1,049,765	2	15,000	Med.	0.996	0.831	0.685	0.48	4	0.986	1	0.975
23	60	1,949,002	1,049,765	2	15,000	Med.	0.996	0.831	0.685	0.48	0.986	1	0.975	0.998
24	1	1,892,194	1,021,458	2	15,000	Med.	0	0	1	0	0.875	0.875	0	0.7
24	5	1,892,194	1,021,458	2	15,000	Med.	0	1	0	0.875	0.7	0.7	0.481	0.481
24	15	1,892,194	1,021,458	2	15,000	Med.	0	0	0.875	0.7	0.481	0.481	0.868	0.7
24	30	1,892,194	1,021,458	2	15,000	Med.	0.998	0.875	0.7	0.481	10	0.99	0.963	0.951
24	60	1,892,194	1,021,458	2	15,000	Med.	1	0.875	0.7	0.481	0.99	0.963	0.978	0.881
25	1	1,740,642	945,620	2	15,000	Med.	0	0.992	0.992	0	0.87	0.87	0.708	0.708
25	5	1,740,642	945,620	2	15,000	Med.	0	0.992	0	0.87	0.708	0.708	0.498	0.498
25	15	1,740,642	945,620	2	15,000	Med.	0.992	0	0.87	0.708	0.498	0.498	0.989	0.708
25	30	1,740,642	945,620	2	15,000	Med.	0.992	0.87	0.708	0.498	11	1	0.953	0.909
25	60	1,740,642	945,620	2	15,000	Med.	0.992	0.87	0.708	0.498	1	0.953	0.909	0.91
26	1	1,955,878	1,087,585	3	10,000	Med.	0	0.944	1	0	0.828	0.828	0.011	0.794
26	5	1,955,878	1,087,585	3	10,000	Med.	0	1	0	0.828	0.794	0.794	0.574	0.574
26	15	1,955,878	1,087,585	3	10,000	Med.	0.944	0	0.828	0.794	0.574	0.574	0.958	0.011
26	30	1,955,878	1,087,585	3	10,000	Med.	0.994	0.828	0.794	0.574	13	0.958	0.956	0.791
26	60	1,955,878	1,087,585	3	10,000	Med.	1	0.828	0.794	0.574	0.958	0.956	0.791	0.896
27	1	1,974,954	1,097,106	3	10,000	Med.	0	1	1	0	0.83	0.83	0	0.795
27	5	1,974,954	1,097,106	3	10,000	Med.	0	1	0.83	0.83	0.795	0.795	0.61	0.61

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Time Elapsed (Min.)	# Vars.	# Cons.	Length of Horizon (Days)	# Samples	Size	NUD60	UD60	UD120	UD240	5 - 0 - UD60	5 - 0 - UD120	5 - 0 - UD240	60 - 1.05 - UD240
27	15	1,974,954	1,097,106	3	10,000	Med.	1	0	0.83	0.795	0.61	0.61	0.982	0
27	30	1,974,954	1,097,106	3	10,000	Med.	1	0.83	0.795	0.61	21	0.982	0.957	0.959
27	60	1,974,954	1,097,106	3	10,000	Med.	1	0.83	0.795	0.61	0.982	0.957	0.959	0.948
28	1	1,997,392	1,108,319	3	10,000	Med.	0	0	1	0	0.812	0.812	0.775	0.775
28	5	1,997,392	1,108,319	3	10,000	Med.	0	1	0.812	0.812	0.775	0.775	0.615	0.615
28	15	1,997,392	1,108,319	3	10,000	Med.	0	0	0.812	0.775	0.615	0.615	0.982	0.775
28	30	1,997,392	1,108,319	3	10,000	Med.	1	0.812	0.775	0.615	9	0.982	0.922	0.82
28	60	1,997,392	1,108,319	3	10,000	Med.	1	0.812	0.775	0.615	0.982	0.922	0.82	0.942
29	1	1,908,742	1,063,943	3	10,000	Med.	0	0.976	1	0	0.84	0.84	0	0.807
29	5	1,908,742	1,063,943	3	10,000	Med.	0	1	0.83	0.84	0.807	0.807	0.632	0.632
29	15	1,908,742	1,063,943	3	10,000	Med.	0.976	0	0.84	0.807	0.632	0.632	0.968	0
29	30	1,908,742	1,063,943	3	10,000	Med.	1	0.84	0.807	0.632	11	0.968	0.976	0.947
29	60	1,908,742	1,063,943	3	10,000	Med.	1	0.84	0.807	0.632	0.968	0.976	0.947	0.886
30	1	2,016,628	1,117,913	3	10,000	Large	0	0.974	1	0	0.822	0.822	0.01	0.784
30	5	2,016,628	1,117,913	3	10,000	Large	0	1	0.822	0.822	0.784	0.784	0.585	0.585
30	15	2,016,628	1,117,913	3	10,000	Large	0.974	0	0.822	0.784	0.585	0.585	0.987	0.01
30	30	2,016,628	1,117,913	3	10,000	Large	1	0.822	0.784	0.585	8	0.987	0.899	0.942
30	60	2,016,628	1,117,913	3	10,000	Large	1	0.822	0.784	0.585	0.987	0.899	0.942	0.951
31	1	304,444	189,749	3	5,000	Small	0.997	0.997	0.997	0.941	0.941	0.941	0.889	0.889
31	5	304,444	189,749	3	5,000	Small	0.997	0.997	0.941	0.941	0.889	0.889	0.61	0.61
31	15	304,444	189,749	3	5,000	Small	0.997	0.941	0.941	0.889	0.61	0.61	0.997	0.998
31	30	304,444	189,749	3	5,000	Small	0.997	0.941	0.889	0.61	0	0.997	0.998	0.997
31	60	304,444	189,749	3	5,000	Small	0.997	0.941	0.889	0.61	0.997	0.998	0.997	1
32	1	353,656	214,302	3	5,000	Small	0.999	0.999	0.999	0.943	0.943	0.943	0.892	0.892
32	5	353,656	214,302	3	5,000	Small	0.999	0.999	0.943	0.943	0.892	0.892	0.581	0.581
32	15	353,656	214,302	3	5,000	Small	0.999	0.943	0.943	0.892	0.581	0.581	0.999	1
32	30	353,656	214,302	3	5,000	Small	0.999	0.943	0.892	0.581	0	0.999	1	0.999
32	60	353,656	214,302	3	5,000	Small	0.999	0.943	0.892	0.581	0.999	1	0.999	0.999
33	1	364,180	219,584	3	5,000	Small	0.998	0.998	0.998	0.872	0.872	0.872	0.824	0.824
33	5	364,180	219,584	3	5,000	Small	0.998	0.998	0.872	0.872	0.824	0.824	0.584	0.584
33	15	364,180	219,584	3	5,000	Small	0.998	0.872	0.872	0.824	0.584	0.584	0.997	0.993
33	30	364,180	219,584	3	5,000	Small	0.998	0.872	0.824	0.584	0	0.997	1	0.997

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Time Elapsed (Min.)	# Vars.	# Cons.	Length of Horizon (Days)	# Samples	Size	NUD60	UD60	UD120	UD240	5 - 0 - UD60	5 - 0 - UD120	5 - 0 - UD240	60 - 1.05 - UD240
33	60	364,180	219,584	3	5,000	Small	0.998	0.872	0.824	0.584	0.997	1	0.997	1
34	1	364,596	219,762	3	5,000	Small	1	1	1	0.932	0.932	0.932	0.887	0.887
34	5	364,596	219,762	3	5,000	Small	1	1	0.932	0.932	0.887	0.887	0.608	0.608
34	15	364,596	219,762	3	5,000	Small	1	0.932	0.932	0.887	0.608	0.608	0.999	0.997
34	30	364,596	219,762	3	5,000	Small	1	0.932	0.887	0.608	0	0.999	0.997	0.996
34	60	364,596	219,762	3	5,000	Small	1	0.932	0.887	0.608	0.999	0.997	0.996	0.997
35	1	344,038	209,449	3	5,000	Small	0.998	0.998	0.998	0.903	0.903	0.903	0.854	0.854
35	5	344,038	209,449	3	5,000	Small	0.998	0.998	0.903	0.903	0.854	0.854	0.612	0.612
35	15	344,038	209,449	3	5,000	Small	0.998	0.903	0.903	0.854	0.612	0.612	1	0.999
35	30	344,038	209,449	3	5,000	Small	0.998	0.903	0.854	0.612	0	1	0.999	1
35	60	344,038	209,449	3	5,000	Small	0.998	0.903	0.854	0.612	1	0.999	1	0.998
36	1	2,709,636	1,500,018	4	10,000	Large	0	0	1	0	0.857	0.857	0	0.827
36	5	2,709,636	1,500,018	4	10,000	Large	0	1	0.857	0.857	0.827	0.827	0.744	0.744
36	15	2,709,636	1,500,018	4	10,000	Large	0	0	0.857	0.827	0.744	0.744	0.997	0
36	30	2,709,636	1,500,018	4	10,000	Large	0.911	0.857	0.827	0.744	40	0.997	0.965	0.721
36	60	2,709,636	1,500,018	4	10,000	Large	1	0.857	0.827	0.744	0.997	0.965	0.727	0.984
37	1	2,276,894	1,283,585	4	10,000	Large	0	0	1	0	0.88	0.88	0	0.85
37	5	2,276,894	1,283,585	4	10,000	Large	0	1	0	0.88	0.85	0.85	0.74	0.74
37	15	2,276,894	1,283,585	4	10,000	Large	0	0	0.88	0.85	0	0.74	0.999	0
37	30	2,276,894	1,283,585	4	10,000	Large	1	0.88	0.85	0	102	0.999	0.995	0.976
37	60	2,276,894	1,283,585	4	10,000	Large	1	0.88	0.85	0.74	0.999	0.995	0.991	0.976
38	1	2,760,022	1,525,133	4	10,000	Large	0	0	1	0	0.805	0.805	0.775	0.775
38	5	2,760,022	1,525,133	4	10,000	Large	0	1	0.801	0.805	0.775	0.775	0.737	0.737
38	15	2,760,022	1,525,133	4	10,000	Large	0	0	0.805	0.775	0.737	0.737	0.98	0.775
38	30	2,760,022	1,525,133	4	10,000	Large	0.97	0.805	0.775	0.737	20	0.98	0.893	0.983
38	60	2,760,022	1,525,133	4	10,000	Large	1	0.805	0.775	0.737	0.98	0.893	0.983	0.983
39	1	2,438,780	1,364,424	4	10,000	Large	0	0.962	1	0	0.856	0.856	0	0.826
39	5	2,438,780	1,364,424	4	10,000	Large	0	1	0.856	0.856	0.826	0.826	0.763	0.763
39	15	2,438,780	1,364,424	4	10,000	Large	0.962	0	0.856	0.826	0.763	0.763	0.998	0.826
39	30	2,438,780	1,364,424	4	10,000	Large	1	0.856	0.826	0.763	34	0.998	0.988	0.972
39	60	2,438,780	1,364,424	4	10,000	Large	1	0.856	0.826	0.763	0.998	0.988	0.972	0.972
40	1	2,537,444	1,413,707	4	10,000	Large	0	0	1	0	0.832	0.832	0.804	0.804

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Time Elapsed (Min.)	# Vars.	# Cons.	Length of Horizon (Days)	# Samples	Size	NUD60	UD60	UD120	UD240	5 - 0 - UD60	5 - 0 - UD120	5 - 0 - UD240	60 - 1.05 - UD240
40	5	2,537,444	1,413,707	4	10,000	Large	0	1	0.832	0.832	0.804	0.804	0.736	0.736
40	15	2,537,444	1,413,707	4	10,000	Large	0	0	0.832	0.804	0.736	0.736	0.976	0.804
40	30	2,537,444	1,413,707	4	10,000	Large	0.963	0.832	0.804	0.736	20	0.976	0.972	0.985
40	60	2,537,444	1,413,707	4	10,000	Large	1	0.832	0.804	0.736	0.976	0.972	0.99	0.987
41	1	1,375,486	831,659	4	5,000	Med.	0	1	1	0.881	0.881	0.881	0.835	0.835
41	5	1,375,486	831,659	4	5,000	Med.	1	1	0.881	0.881	0.835	0.835	0.806	0.806
41	15	1,375,486	831,659	4	5,000	Med.	1	0.881	0.881	0.835	0.806	0.806	0.998	0.983
41	30	1,375,486	831,659	4	5,000	Med.	1	0.881	0.835	0.806	6	0.998	0.999	0.999
41	60	1,375,486	831,659	4	5,000	Med.	1	0.881	0.835	0.806	0.998	0.999	0.999	1
42	1	1,315,546	801,605	4	5,000	Med.	0	1	1	0.882	0.882	0.882	0.84	0.84
42	5	1,315,546	801,605	4	5,000	Med.	1	1	0.882	0.882	0.84	0.84	0.814	0.814
42	15	1,315,546	801,605	4	5,000	Med.	1	0.882	0.882	0.84	0.814	0.814	0.998	0.995
42	30	1,315,546	801,605	4	5,000	Med.	1	0.882	0.84	0.814	4	0.998	1	1
42	60	1,315,546	801,605	4	5,000	Med.	1	0.882	0.84	0.814	0.998	1	1	1
43	1	1,376,592	832,173	4	5,000	Med.	0.988	1	1	0	0.862	0.862	0.813	0.813
43	5	1,376,592	832,173	4	5,000	Med.	1	1	0.862	0.862	0.813	0.813	0.778	0.778
43	15	1,376,592	832,173	4	5,000	Med.	1	0	0.862	0.813	0.778	0.778	0.999	0.951
43	30	1,376,592	832,173	4	5,000	Med.	1	0.862	0.813	0.778	8	0.999	0.999	1
43	60	1,376,592	832,173	4	5,000	Med.	1	0.862	0.813	0.778	0.999	0.999	1	1
44	1	1,306,294	797,005	4	5,000	Med.	0.998	0.998	0.998	0.88	0.88	0.88	0.835	0.835
44	5	1,306,294	797,005	4	5,000	Med.	0.998	0.998	0.88	0.88	0.835	0.835	0.808	0.808
44	15	1,306,294	797,005	4	5,000	Med.	0.998	0.88	0.88	0.835	0.808	0.808	1	0.981
44	30	1,306,294	797,005	4	5,000	Med.	0.998	0.88	0.835	0.808	5	1	1	1
44	60	1,306,294	797,005	4	5,000	Med.	0.998	0.88	0.835	0.808	1	1	1	0.998
45	1	1,252,562	770,184	4	5,000	Med.	0	1	1	0	0.928	0.928	0.88	0.88
45	5	1,252,562	770,184	4	5,000	Med.	1	1	0.928	0.928	0.88	0.88	0.854	0.854
45	15	1,252,562	770,184	4	5,000	Med.	1	0	0.928	0.88	0.854	0.854	1	0.98
45	30	1,252,562	770,184	4	5,000	Med.	1	0.928	0.88	0.854	7	1	0.996	0.996
45	60	1,252,562	770,184	4	5,000	Med.	1	0.928	0.88	0.854	1	0.996	0.996	1
46	1	3,305,890	1,833,463	5	10,000	Large	0	0	0.931	0	0.848	0.848	0.816	0.816
46	5	3,305,890	1,833,463	5	10,000	Large	0	0.931	0	0.848	0.816	0.816	0.791	0.791
46	15	3,305,890	1,833,463	5	10,000	Large	0	0	0.848	0.816	0.791	0.791	1	0.816

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Time Elapsed (Min.)	# Vars.	# Cons.	Length of Horizon (Days)	# Samples	Size	NUD60	UD60	UD120	UD240	5 - 0 - UD60	5 - 0 - UD120	5 - 0 - UD240	60 - 1.05 - UD240
46	30	3,305,890	1,833,463	5	10,000	Large	0	0.848	0.816	0.791	50	1	0.971	0.726
46	60	3,305,890	1,833,463	5	10,000	Large	0.931	0.848	0.816	0.791	1	0.971	0.726	0.988
47	1	3,234,804	1,798,044	5	10,000	Large	0	0	0.998	0	0.851	0.851	0	0.815
47	5	3,234,804	1,798,044	5	10,000	Large	0	0.998	0	0.851	0.815	0.815	0.787	0.787
47	15	3,234,804	1,798,044	5	10,000	Large	0	0	0.851	0.815	0.787	0.787	0.997	0
47	30	3,234,804	1,798,044	5	10,000	Large	0.979	0.851	0.815	0.787	48	0.997	1	0.698
47	60	3,234,804	1,798,044	5	10,000	Large	0.998	0.851	0.815	0.787	0.997	1	0.698	0.998
48	1	3,399,906	1,880,512	5	10,000	Large	0	0	1	0	0.836	0.836	0.799	0.799
48	5	3,399,906	1,880,512	5	10,000	Large	0	1	0.834	0.836	0.799	0.799	0.772	0.772
48	15	3,399,906	1,880,512	5	10,000	Large	0	0	0.836	0.799	0.772	0.772	0.997	0.799
48	30	3,399,906	1,880,512	5	10,000	Large	0	0.836	0.799	0.772	15	0.997	0.974	0.982
48	60	3,399,906	1,880,512	5	10,000	Large	1	0.836	0.799	0.772	0.997	0.974	0.982	0.982
49	1	3,192,728	1,776,841	5	10,000	Large	0	0	0.998	0	0.841	0.841	0	0.808
49	5	3,192,728	1,776,841	5	10,000	Large	0	0.998	0	0.841	0.808	0.808	0.77	0.77
49	15	3,192,728	1,776,841	5	10,000	Large	0	0	0.841	0.808	0.77	0.77	1	0.808
49	30	3,192,728	1,776,841	5	10,000	Large	0.926	0.841	0.808	0.77	49	1	0.996	0.7
49	60	3,192,728	1,776,841	5	10,000	Large	0.998	0.841	0.808	0.77	1	0.996	0.7	0.974
50	1	3,339,016	1,850,121	5	10,000	Large	0	0	0.95	0	0.858	0.858	0	0.822
50	5	3,339,016	1,850,121	5	10,000	Large	0	0.95	0	0.858	0.822	0.822	0.793	0.793
50	15	3,339,016	1,850,121	5	10,000	Large	0	0	0.858	0.822	0.793	0.793	0.983	0
50	30	3,339,016	1,850,121	5	10,000	Large	0	0.858	0.822	0.793	37	0.983	0.985	1
50	60	3,339,016	1,850,121	5	10,000	Large	0.95	0.858	0.822	0.793	0.983	0.985	1	1
51	1	1,743,006	1,050,816	5	5,000	Med.	0	1	1	0.879	0.879	0.879	0.815	0.815
51	5	1,743,006	1,050,816	5	5,000	Med.	0	1	0.879	0.879	0.815	0.815	0.778	0.778
51	15	1,743,006	1,050,816	5	5,000	Med.	1	0.879	0.879	0.815	0.778	0.778	0.995	0.994
51	30	1,743,006	1,050,816	5	5,000	Med.	1	0.879	0.815	0.778	3	0.995	0.999	1
51	60	1,743,006	1,050,816	5	5,000	Med.	1	0.879	0.815	0.778	0.995	0.999	1	0.992
52	1	1,657,794	1,008,232	5	5,000	Med.	0	1	1	0	0.896	0.896	0.84	0.84
52	5	1,657,794	1,008,232	5	5,000	Med.	1	1	0.896	0.896	0.84	0.84	0.809	0.809
52	15	1,657,794	1,008,232	5	5,000	Med.	1	0	0.896	0.84	0.809	0.809	0.997	0.998
52	30	1,657,794	1,008,232	5	5,000	Med.	1	0.896	0.84	0.809	3	0.997	0.998	0.99
52	60	1,657,794	1,008,232	5	5,000	Med.	1	0.896	0.84	0.809	0.997	0.998	0.99	1

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Time Elapsed (Min.)	# Vars.	# Cons.	Length of Horizon (Days)	# Samples	Size	NUD60	UD60	UD120	UD240	5 - 0 - UD60	5 - 0 - UD120	5 - 0 - UD240	60 - 1.05 - UD240
53	1	1,832,988	1,095,905	5	5,000	Med.	0	0.996	0.996	0.908	0.908	0.908	0.85	0.85
53	5	1,832,988	1,095,905	5	5,000	Med.	0	0.996	0.908	0.908	0.85	0.85	0.818	0.818
53	15	1,832,988	1,095,905	5	5,000	Med.	0.996	0.908	0.908	0.85	0.818	0.818	1	0.982
53	30	1,832,988	1,095,905	5	5,000	Med.	0.996	0.908	0.85	0.818	3	1	0.986	0.98
53	60	1,832,988	1,095,905	5	5,000	Med.	0.996	0.908	0.85	0.818	1	0.986	0.98	0.979
54	1	1,665,390	1,012,064	5	5,000	Med.	0	1	1	0	0.937	0.937	0.877	0.877
54	5	1,665,390	1,012,064	5	5,000	Med.	1	1	0.937	0.937	0.877	0.877	0.841	0.841
54	15	1,665,390	1,012,064	5	5,000	Med.	1	0	0.937	0.877	0.841	0.841	0.996	0.994
54	30	1,665,390	1,012,064	5	5,000	Med.	1	0.937	0.877	0.841	4	0.996	1	0.991
54	60	1,665,390	1,012,064	5	5,000	Med.	1	0.937	0.877	0.841	0.996	1	0.991	0.991
55	1	1,734,194	1,046,389	5	5,000	Med.	0	0.999	0.999	0.889	0.889	0.889	0.833	0.833
55	5	1,734,194	1,046,389	5	5,000	Med.	0.992	0.999	0.889	0.889	0.833	0.833	0.804	0.804
55	15	1,734,194	1,046,389	5	5,000	Med.	0.999	0.889	0.889	0.833	0.804	0.804	0.999	1
55	30	1,734,194	1,046,389	5	5,000	Med.	0.999	0.889	0.833	0.804	2	0.999	1	0.999
55	60	1,734,194	1,046,389	5	5,000	Med.	0.999	0.889	0.833	0.804	0.999	1	0.999	0.999
56	1	2,452,604	1,476,603	7	5,000	Large	0	0	0.998	0	0.901	0.901	0.833	0.833
56	5	2,452,604	1,476,603	7	5,000	Large	0	0.998	0.901	0.901	0.833	0.833	0.794	0.794
56	15	2,452,604	1,476,603	7	5,000	Large	0	0	0.901	0.833	0.794	0.794	1	0.833
56	30	2,452,604	1,476,603	7	5,000	Large	0.998	0.901	0.833	0.794	7	1	0.996	0.998
56	60	2,452,604	1,476,603	7	5,000	Large	0.998	0.901	0.833	0.794	1	0.996	0.998	0.998
57	1	2,667,758	1,584,269	7	5,000	Large	0	0	1	0	0.955	0.955	0.877	0.877
57	5	2,667,758	1,584,269	7	5,000	Large	0	1	0.955	0.955	0.877	0.877	0.835	0.835
57	15	2,667,758	1,584,269	7	5,000	Large	0	0	0.955	0.877	0.835	0.835	0.995	0.877
57	30	2,667,758	1,584,269	7	5,000	Large	1	0.955	0.877	0.835	7	0.995	0.994	1
57	60	2,667,758	1,584,269	7	5,000	Large	1	0.955	0.877	0.835	0.995	0.994	1	1
58	1	2,082,018	1,291,228	7	5,000	Large	0	1	1	0	0.977	0.977	0.901	0.901
58	5	2,082,018	1,291,228	7	5,000	Large	0	1	0.977	0.977	0.901	0.901	0.864	0.864
58	15	2,082,018	1,291,228	7	5,000	Large	1	0	0.977	0.901	0.864	0.864	0.998	1
58	30	2,082,018	1,291,228	7	5,000	Large	1	0.977	0.901	0.864	13	0.998	1	0.996
58	60	2,082,018	1,291,228	7	5,000	Large	1	0.977	0.901	0.864	0.998	1	0.996	0.996
59	1	2,078,980	1,289,713	7	5,000	Large	0	0.996	0.996	0	0.996	0.996	0	0.936
59	5	2,078,980	1,289,713	7	5,000	Large	0	0.996	0	0.996	0.936	0.936	0.892	0.892

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Time Elapsed (Min.)	# Vars.	# Cons.	Length of Horizon (Days)	# Samples	Size	NUD60	UD60	UD120	UD240	5 - 0 - UD60	5 - 0 - UD120	5 - 0 - UD240	60 - 1.05 - UD240
59	15	2,078,980	1,289,713	7	5,000	Large	0.996	0	0.996	0.936	0.892	0.892	0.996	0.936
59	30	2,078,980	1,289,713	7	5,000	Large	0.996	0.996	0.936	0.892	34	0.996	1	0.997
59	60	2,078,980	1,289,713	7	5,000	Large	0.996	0.996	0.936	0.892	0.996	1	0.997	0.997
60	1	2,515,326	1,508,049	7	5,000	Large	0	0	1	0	0.956	0.956	0	0.88
60	5	2,515,326	1,508,049	7	5,000	Large	0	1	0	0.956	0.88	0.88	0.842	0.842
60	15	2,515,326	1,508,049	7	5,000	Large	0	0	0.956	0.88	0.842	0.842	0.996	0.88
60	30	2,515,326	1,508,049	7	5,000	Large	0.979	0.956	0.88	0.842	55	0.996	0.996	0.997
60	60	2,515,326	1,508,049	7	5,000	Large	1	0.956	0.88	0.842	0.996	0.996	0.997	0.997