A computational study of continuous and discrete time formulations for a class of short-term scheduling problems for multipurpose plants

Saman Lagzi,[†,‡] Do Yeon Lee,[†] Ricardo Fukasawa,[†] and Luis Ricardez-Sandoval[*,†]

[†]*University of Waterloo, Waterloo, Canada*

[‡]*Current affiliation: Rotman School of Business, University of Toronto*

E-mail: laricardezsandoval@uwaterloo.ca(*)

Abstract

A key decision in scheduling problems is deciding when to perform certain operations and the quality of solutions depends on how time is represented. The two main classes of time representation are discrete-time approaches (with uniform or non-uniform discretization schemes) and continuous-time approaches. In this work, we compare the performance of these two classes for short-term scheduling of multipurpose facilities with single purpose machines, constant processing times, discrete batches, material splitting, multitasking, and no batch mixing. In addition, the different discretization schemes were compared against each other. We show that, for the modeling framework proposed in this work, the selected discrete-time formulation typically obtained higher quality solutions, and required less time to solve compared to the selected continuous-time formulation, as the continuous-time formulation exhibited detrimental trade-off between computational time and solution quality. We also show that within the scope of this study, non-uniform discretization schemes typically yielded solutions

of similar quality compared to a fine uniform discretization scheme, but required only a fraction of the computational time. A total of 190 small and industrial-sized instances, comprised of 1,030 runs were considered for this study.

1 Introduction

Scheduling problems are a very important class of optimization problems in the process systems engineering (PSE) community, which involves determining the timings of various operations or tasks subject to some operational constraints. Such decisions are affected by the way time is represented in the scheduling model. The two most common approaches for time representations are: discrete-time approaches, which prespecify a finite set of time points at which scheduling decisions may occur, and continuous-time approaches, in which the optimization model determines the positions of a specified number of time points through the use of continuous decision variables. Each of these choices may have significant effects on both the quality of the schedules and the time needed to obtain these schedules. This work aims to study the effects of these choices, and to do so, we have compared the performances of the state of the art discrete and continuous-time formulations in the context of short-term scheduling of operations in a multipurpose plant.

To properly introduce some terminology, we start by describing a particular application of our work – short-term scheduling in a multipurpose plant. However, the approaches presented in this study can be applied to other case studies with similar characteristics. A multipurpose plant receives a set of tasks, where each task, composed of a specific number of samples (i.e., batches are discrete), needs to visit a specific sequence of processing units, called a path. Different tasks may have different paths. Each processing unit is composed of a set of machines; a sample belonging to a task needs to be processed at one of the machines in a processing unit, before it can proceed to the next processing unit in its path. Machines have specified capacities and processing times and are assumed to run continuously without

interruptions (i.e., batch mixing is not allowed). Accordingly, samples being processed in a machine cannot be removed from the machine while it is in operation. In this work, we deal with problems where samples in a task may be split, that is, it is possible for just a subset of samples of a task to be processed in any given machine. This feature is called *material splitting*. In addition, each machine has the ability to *multitask*, that is, it may process samples from several different tasks simultaneously. Each machine can only perform one process, and the machines have constant and identical processing times for all compatible tasks.

In this work, we solve problems based on an actual multipurpose plant in the analytical services sector. This is a major sector in which various types of analyses are carried out on a set of samples in order to determine its properties and chemical composition, which can be used in the decision making process by end-customers in various industries such as mining, healthcare, oil & gas, and food. Such facilities may receive samples in the order of thousands to be processed on a daily basis. Therefore, devising efficient scheduling algorithms for such facilities is both challenging and economically important.

Over the past few decades, several discrete-time formulations have been proposed, in which scheduling decisions, such as the beginning of sample processing by a machine, can only occur on a finite set of prespecified time points. A general discrete-time Mixed Integer Linear Program (MILP) algorithm for short-term scheduling of batch operations in a multiproduct/multipurpose plant was proposed by Kondili et al.[1], which included batch operations and material/products explicitly as network nodes in a *State-Task Network (STN)*. Computational issues surrounding this approach were discussed by Shah et al.[2]; Pantelides[3] presented the more general *Resource-Task Network (RTN)* representation as a unified framework. The STN representation is still widely used today. Patil et al.[4] presented an Integer Program (IP) based on a uniform discrete-time formulation utilizing the STN representation,

which included the multitasking feature.

Traditionally, as Velez and Maravelias [5] have noted, discrete-time models only represented time on a single grid with uniformly spaced time points where events may occur. [1–4,6] These uniform discrete-time formulations could result in MILPs involving large numbers of binary variables, leading to large computational demands. [1,2] To address this issue, Velez and Maravelias [5] proposed a flexible discrete-time formulation with multiple and possibly non-uniform time grids. In a follow up study, Velez and Maravelias [7] generalized their ideas into a general framework for developing such scheduling models, and further explored the algorithms for generating the non-uniform time grids. Using this approach, events can happen at different times for different machines, effectively allowing many decisions to be made for some of the machines without defining unnecessary time points for the rest of the machines. Such an approach has the advantage of obtaining a better balance between solution quality and computational demand, since finer discretizations are only used when needed, for example, for processing units with relatively short processing times. Velez et al. [8] successfully implemented the multi-grid approach to solve three large examples, including a case study based on production operations by the Dow Chemical Company, while extending the approach to consider various features including changeovers and storage policies.

One important aspect of the discrete-time formulations is that a very fine discretization (e.g., every minute) allows for a high degree of flexibility in the solutions, possibly resulting in higher quality solutions compared to a coarser discretization. However, such a fine discretization requires a larger number of decision variables, and results in larger (and more computationally intensive) optimization problems to solve. Thus, there exists a trade-off between computational time and solution quality.

A different approach only predefines the number of time points, and the optimization

algorithm determines the positions of the time points (i.e., the times within the scheduling horizon at which scheduling decisions may occur). The formulations implementing this approach are referred to as continuous-time formulations. Events may either occur on a single global time scale for all units,[9–20] or on multiple unit-specific time grids.[21–28] Since the timing of events is not predetermined in continuous-time formulations, more accurate solutions may be obtained without considerably increasing the size of the formulations. Furthermore, the continuous-time formulations are capable of modeling some operational features, such as variable machine processing times, with relative ease, which are challenging for discrete-time formulations. The flexibility provided by continuous-time approaches also make them suitable for developing frameworks for scheduling under uncertainty, as Lappas and Gounaris[29] have demonstrated by successfully developing an adjustable robust optimization (ARO) framework utilizing the global event-based model by Castro et al.[13], and Wang et al.[30] developed a rescheduling framework for crude-oil operations under uncertainty utilizing the unit-specific model by Furman et al.[26].

While discrete-time and continuous-time formulations have often been studied separately, few studies have considered a comparison between these two approaches.[31–33] Stefansson et al.[31] compared these two approaches with a case study based on a multistage, multiproduct pharmaceutical production facility. They solved their strongly NP-hard problem by decomposing the problem into two parts, and solving the individual parts using discrete-time and continuous-time formulations. Stefansson et al.[31] found the continuous-time approach to be more suitable in solving their problem as it required less time to solve, and the resulting schedules were more precise. Sundaramoorthy and Maravelias[32] compared these two approaches using a collection of more than 100 problem instances. They found that while continuous-time formulations appeared to solve faster than their discrete-time counterparts for relatively short scheduling horizons, the performance of the continuous-time formulations suffered significantly with increasing length of horizons and increasing number of time points.

On the other hand, the performance of discrete-time formulations was found to be relatively consistent. Sundaramoorthy and Maravelias[32] also found the discrete-time formulations to consistently provide better solutions. Merchan et al.[33] compared four discrete-time models against a continuous-time model using a total of 1808 runs, and found that all the proposed discrete-time models outperformed the continuous-time model as the discrete-time models required significantly less time to solve.

Nevertheless, the aforementioned comparison studies either did not consider the situation where the discrete-time formulation is capable of accommodating flexible time discretization,[31,32] or the continuous-time formulation is unable to handle material splitting[31,33] or multitasking.[31–33] In this work, we consider small and industrial-sized instances with discrete batches, material splitting, and multitasking, where batches are not allowed to mix, and the machines are single purpose machines with constant and identical processing times for all compatible tasks. Based on the above, the main contributions of this study are as follows:

- We extend the discrete-time formulation of Patil et al.[4] to incorporate the flexible discrete-time formulation approach of Velez and Maravelias[5]. The model of Patil et al.[4] was selected for extension as it is able to handle multitasking and is well suited to handle the features of an analytical services facility.

- We perform a comparison between the performance of the flexible discrete-time formulation with the performance of a modified version of the continuous-time formulation presented by Lagzi et al.[20], which is able to handle material splitting and multitasking. To the best of our knowledge, this is the first such comparison in the literature.

- Through a series of computational experiments, we aim to study the strengths and weaknesses of the different time representation approaches relative to each other, and by doing so, determine the favorable approach to solving industrial-sized short-term

scheduling problems for a multipurpose plant.

The rest of the study is organized as follows: In Section 2, we present a detailed description of the problem along with both the flexible discrete-time and continuous-time formulations for the problem. Computational results and discussion are presented in Section 3; the conclusion and possible future developments are discussed in Section 4.

2 Problem Definition and Formulations

The problem that will be considered in this study is based on an actual analytical services facility. The facility receives a set of tasks, I, and needs to process them within a scheduling horizon, H, using a set of processing units, P. Each processing unit, $p \in P$, consists of a set of identical machines, J_p, that perform a specific process. The set of all machines in the facility is denoted by J, where J is the disjoint union of J_p's.

For task $i \in I$, a_{ik} number of samples become available at the beginning of the scheduling horizon for process $p^i_k \in P$, which need to visit a specific sequence of processing units, called a path. The path of tasks i, denoted by S_i, is a sequence of $n(i)$ distinct processing units $\{p^i_1, \ldots, p^i_{n(i)}\}$, where $p^i_k \in P$ for all $i \in I$, $k = 1, \ldots, n(i)$. Within the context of this work, samples for a task may be introduced at any processing unit in the path at the beginning of the scheduling horizon. The incoming samples $a_{ik}, \forall 1 < k \leq n(i)$ simulate samples that had been processed at the preceding processing unit p^i_{k-1} during a previous scheduling horizon. The samples in task i introduced at p^i_k must visit each subsequent processing unit in S_i sequentially, that is, p^i_{k+1} in S_i can only be visited if p^i_k has already been visited. Samples are considered to have visited processing unit, p, if it has been processed by one of the machines in J_p. It is assumed that at the beginning of the scheduling horizon, the machines in the facility, $j \in J$, will become available to start processing materials at the facility. We also assume that there is no transportation time between different processing units, and that

there are no restrictions on intermediate storage.

Machines in a processing unit p have a specific capacity, denoted as β_p, and an associated processing time, denoted as $\tau(p)$. This means that machine $j \in J_p$ can be loaded with at most β_p number of samples from potentially different tasks. Once a machine has been turned on to process the samples, it will run without interruption for a time $\tau(p)$. After this time, the machine is considered to be available; also, the samples are considered to have visited the corresponding processing unit and they are ready to visit the next processing unit in their path. Since machines are assumed to be identical, the processing time of the machines in a processing unit can be referred to as the processing time of the processing unit. Furthermore, there is no minimum working capacity for any machine, that is, the machines can be turned on with any number of samples between 0 and β_p. It is assumed that the information described above is available and known *a priori*.

For the facility under consideration, maximization of the number of samples to be processed is a key priority. Ideally, we would like to maximize the true throughput of the facility over an extended period of time, defining the throughput to be the number of samples for which the last process in their respective paths have been completed. However, we can only define the objective function over the scheduling horizon, and defining the objective function to maximize the throughput can only lead to suboptimal resource utilization over an extended period of time in a capacity constrained environment. This suboptimal resource utilization happens when there is not enough time or capacity to finish processing some samples within the scheduling horizon, and a simple throughput maximization objective would have no objective incentive to start processing these samples. With such an objective function, processing units with processing times that are longer than the scheduling horizon will always be idle. Therefore, the objective functions to be presented for the discrete and continuous-time formulations in Sections 2.1 and 2.2 will give incentive for samples to be

processed even if they cannot reach the end of their respective paths, and even if processing cannot be completed within the scheduling horizon. Similarly, flow constraints for these formulations will allow processes to continue processing even after the end of the scheduling horizon, once processing starts for a machine. Processing may also start at the end of the scheduling horizon, but no processes are allowed to start after the end of the scheduling horizon. This particular feature in our formulation is indeed required to reflect the actual operation in analytical service facilities.

2.1 Flexible discrete-time formulation

The flexible discrete-time formulation presented in this study extends the multitasking uniform discrete-time IP formulation presented by Patil et al.[4], by incorporating the multiple non-uniform time grids approach proposed by Velez and Maravelias[5]. To derive such extension, additional notation and assumptions are needed. The revised formulation is presented below. The time domain for each individual processing unit will be discretized into a predetermined series of time points.

Each $p \in P$ will have a time step $\Delta(p)$, which represents the time elapsed between two consecutive time points for processing unit p. Let $\mathcal{E}(p) = (0, \varphi_{p1}, \varphi_{p2}, \ldots, \varphi_{p(|\mathcal{E}(p)|-2)}, H)$ represent the increasing sequence of time points of processing unit p along the axis of time. Let $\varphi_{pt}, \forall t = 0, \ldots, |\mathcal{E}(p)| - 1$ represent the t^{th} element of $\mathcal{E}(p)$, i.e., the time value of the t^{th} time point of a discretized time grid. Given the above descriptions for the flexible discrete-time representation, we define a time step as

$$\Delta(p) := \varphi_{p(t+1)} - \varphi_{pt};\ \forall t = 1, \ldots, |\mathcal{E}(p)| - 2 \tag{1}$$

Note that, $\forall\ p \in P$, $\mathcal{E}(p)$ is known *a priori*; also the machines in the processing unit can only

be turned on at the beginning of a time point of the processing unit. The present formulation assumes that the time steps are fixed for each processing unit; however, this condition can be relaxed to consider variable time steps. The different discretization schemes to be considered for this work will be discussed in Section 3.1.

Since the machines in a processing unit are assumed to be identical in the flexible discrete-time formulation, it is not needed to consider the machines J_p of processing unit p on an individual basis, rather they are considered as resources of the processing units. Accordingly, $R_p := |J_p|$ represents the number of machines in processing unit p, for all $p \in P$.

The first set of decision variables for the flexible discrete-time formulation are described next.

B_{ikt}: Is a nonnegative integer variable representing the number of samples from task i that are set to start being processed at the k^{th} processing unit in the path of task i, $p_k^i \in S_i$, at the time point t, $\forall\ i \in I,\ 1 \leq k \leq n(i),\ t = 0, \ldots, |\mathcal{E}(p_k^i)| - 1$.

X_{pt}: Is a nonnegative integer variable representing the number of machines from processing unit p that are being used at time point t, $\forall\ p \in P,\ t = 0, \ldots, |\mathcal{E}(p)| - 1$.

Constraints (2)-(3) ensure that the machines in the facility are not overloaded with samples.

$$\sum_{i,k:p=p_k^i} B_{ikt} \leq\ X_{pt}\beta_p; \quad \forall\ p \in P,\ t = 0, \ldots, |\mathcal{E}(p)| - 1 \tag{2}$$

$$\sum_{\theta \in \mathcal{E}(p):\varphi_{pt} < \varphi_{p\theta} + \tau(p) \leq \varphi_{pt} + \tau(p)} X_{p\theta} \leq R_p; \quad \forall\ p \in P,\ t = 0, \ldots, |\mathcal{E}(p)| - 1 \tag{3}$$

As mentioned above, a sample from task i can visit processing unit p_k^i in S_i only if it has already visited the previous processing unit, p_{k-1}^i, in S_i. Also, a sample from task i is considered to have visited processing unit p_k^i in S_i, if it has been processed by one of the machines in J_p. To account for these conditions, the following notation is considered:

W_{ikt}: is a nonnegative integer variable representing the number of samples from task i that have visited p^i_{k-1} and are ready to visit processing unit p^i_k at time point t , $\forall\ i \in I,\ k = 2, \ldots, n(i),\ t = 0, \ldots, |\mathcal{E}(p^i_k)| - 1$.

Constraint (4) ensures that a subset of materials from task i can visit processing unit p^i_k in S_i at time point t, if it has already visited processing unit p^i_{k-1} in S_i before time point t. Furthermore, it ensures that the same number of samples from task i that enter a processing unit, leave the processing unit after completing their processing, i.e., constraint (4) is a flow conservation constraint, preventing samples from being created or lost. Because of its structure, constraint (4) can not be extended to the first processing unit in the path of a task and the first time point of the processing unit; hence, constraint (5) is introduced to extend constraint (4) to the first process in the paths of the tasks.

$$B_{ikt} + W_{ikt} = W_{ik(t-1)} + \sum_{\substack{\theta=1,\ldots,|\mathcal{E}(p^i_{k-1})|: \\ \varphi_{p^i_k t-1} < \varphi_{p^i_{k-1}\theta} + \tau(p^i_{k-1}) \leq \varphi_{p^i_k t}}} B_{i(k-1)\theta}; \tag{4}$$

$$\forall\ i \in I,\ k = 2, \ldots, n(i),\ t = 1, \ldots, |\mathcal{E}(p^i_k)| - 1$$

$$B_{i1t} + W_{i1t} = W_{i1(t-1)};\quad \forall\ i \in I,\ t = 1, \ldots, |\mathcal{E}(p^i_k)| - 1 \tag{5}$$

Constraint (6) introduces a_{ik} samples for task i at processing unit p^i_k at the beginning of the scheduling horizon as the number of samples waiting to start processing at that unit.

$$W_{ik0} = a_{ik};\quad \forall\ i \in I,\ k = 1 \ldots n(i) \tag{6}$$

Given that the samples are introduced as samples waiting to start processing at the first time point, constraint (7) prevents samples from being processed at the first time point. Therefore, for each discretization scheme and for all processing units, discretization was carried out to set the value of the first two time points, φ_{p0} and φ_{p1}, to zero in order to allow events to occur at the beginning of the scheduling horizon. For this reason, Equation (1)

defined the time steps starting at $t = 1$ rather than $t = 0$.

$$B_{ik0} = 0; \quad \forall\ i \in I,\ k = 1 \ldots n(i) \tag{7}$$

The following objective function aims to maximize throughput while optimally utilizing available resources.

$$maximize \sum_{i \in I} \sum_{k=1}^{n(i)} \sum_{t=0}^{|\mathcal{E}(p_k^i)|-1} \frac{k}{n(i)} B_{ikt} \tag{8}$$

The weight $\frac{k}{n(i)}$ is the relative position of processing unit p_k^i in path S_i with $n(i)$ number of processing units. For example, the 4^{th} processing unit (p_4^i) in a path with 7 processing units ($n(i) = 7$) will have a weight of $\frac{4}{7}$. This weight $\frac{k}{n(i)}$, therefore, gives priority to beginning the last process of a task's path.

2.2 Continuous-time formulation

Lagzi et al.[20] introduced a global event-based MILP formulation that has been modified to model the problem considered in this work. For the purpose of completeness, we will describe the formulation here as used in this work. The formulation was modified to remove features that were not considered for this work, namely, minimum processing load, and the feature allowing samples to be introduced after the beginning of the scheduling horizon. An *idle* task will be assigned to a machine whenever the machine is being idle. The idle task is denoted as task number 0 whereas $I = 1, ..., |I|$ is the set of the actual tasks. Note that, for every $p \in P$ and $j \in J_p$, we denote by I_j, the subset of tasks $i \in I$ where $p \in S_i$.

The time domain will be partitioned into $N + 1$ predetermined number of time points, where the distance between two consecutive time points is called a time slot. The time points are shared by all the tasks and machines and their locations along the time axis is determined through the optimization model. Note that the first and last time points, 0 and N, have their locations fixed at the beginning and at the end of the scheduling horizon, i.e. times 0 and H, respectively. Accordingly, $T_n \in [0, H],\ \forall\ n = 0, \ldots, N$, is the decision

variable representing the location of time point n and $SL_n \in [0, H]$ is the decision variable representing the length of time slot n, where time slot n is the time between T_n and T_{n-1}, for all $n \in 1, \ldots, N$. Notice that the number of time points is assumed to be given as an input. The problem of determining the appropriate value of N will be discussed in Section 3.1. The rest of the decision variables used in the continuous-time formulation are as follows:

$Z_{jn} \in \{0, 1\}$: 1 if machine j is turned on at time point n, and 0 otherwise; $\forall\ j \in J,\ \ n = 0, \ldots, N$.

$Y_{ijn} \in \{0, 1\}$: 1 if samples from task i start being processed at machine j at time point n, and 0 otherwise; $\forall\ j \in J,\ i \in I \cup \{0\},\ n = 0, \ldots, N$.

$YE_{ijn} \in \{0, 1\}$: 1 if machine j is set to finish processing samples from task i at time point n, and 0 otherwise; $\forall\ j \in J,\ i \in I \cup \{0\},\ n = 0, \ldots, N$.

$YR_{ijn} \in \{0, 1\}$: 1 if machine j is set to continue processing samples from task i at time point n, and 0 otherwise; $\forall\ j \in J,\ i \in I \cup \{0\},\ n = 0, \ldots, N$.

TR_{ijn}: A nonnegative continuous variable, representing the amount of time remaining to complete processing samples from task i that are set to continue being processed at machine j at time point n; $\forall\ j \in J,\ i \in I_j \cup \{0\},\ n = 0, \ldots, N$.

B_{ijn}: A nonnegative integer variable representing the number of samples from task i that are set to begin processing at machine j at time point n; $\forall\ j \in J,\ i \in I_j \cup \{0\},\ n = 0, \ldots, N$.

BE_{ijn}: A nonnegative integer variable representing the number of samples from task i that are set to finish being processed at machine j at time point n; $\forall\ j \in J,\ i \in I_j \cup \{0\},\ n = 0, \ldots, N$.

BR_{ijn}: A nonnegative integer variable representing the number of samples from task i that are set to continue being processed at machine j at time point n; $\forall\ j \in J,\ i \in I_j \cup \{0\},\ n = 0, \ldots, N$.

W_{ikn}: A nonnegative integer variable representing the number of samples from task i that have visited p^i_{k-1} and are ready to visit processing unit p^i_k at time point n; $\forall\ i \in I \cup \{0\},\ k = 2, \ldots, n(i),\ n = 0, \ldots, N$.

Constraints (9)-(11) model the relationship between T_n and SL_n.

$$\sum_{n=1}^{N} SL_n = H \tag{9}$$

$$T_n - T_{n-1} = SL_n; \;\; \forall\, n = 1, \ldots, N \tag{10}$$

$$T_0 = 0 \tag{11}$$

Constraints (12)-(18) ensure that turning machine j on at time point n is coordinated with machine j starting, continuing, or finishing processing samples from a set of tasks. They also assure that a machine cannot continue processing samples and start processing new samples, at any time point.

$$Z_{jn} \geq \; Y_{ijn}; \;\; \forall\, j \in J,\; i \in I_j \cup \{0\},\; n = 0, \ldots, N \tag{12}$$

$$Z_{jn} \leq \sum_{i \in I_j \cup \{0\}} Y_{ijn}; \;\; \forall\, j \in J,\; n = 0, \ldots, N \tag{13}$$

$$Z_{jn} \geq YE_{ijn}; \;\; \forall\, j \in J,\; i \in I_j \cup \{0\},\; n = 1 \ldots, N \tag{14}$$

$$\sum_{i \in J} \sum_{i \in I_j \cup \{0\}} YE_{ij0} = 0 \tag{15}$$

$$Z_{jn} \leq 1 - YR_{ijn}; \;\; \forall\, j \in J,\; i \in I_j \cup \{0\},\; n = 0, \ldots, N \tag{16}$$

$$(1 - \sum_{i \in I_j \cup \{0\}} YR_{ijn}) \leq Z_{jn}; \;\; \forall\, j \in J,\; n = 1, \ldots, N \tag{17}$$

$$\sum_{i \in I_j \cup \{0\}} YR_{ij0} = 0; \;\; \forall\, j \in J \tag{18}$$

Constraint (19) ensures that, if samples from task i started or continued to be processed at machine j at time point n, then, at the next time point the samples will either complete their processing or continue being processed in machine j.

$$YR_{ijn} = YR_{ij(n-1)} + Y_{ij(n-1)} - YE_{ijn}; \;\; \forall\, j \in J,\; i \in I_j \cup \{0\},\; n = 1, \ldots, N \tag{19}$$

Constraint (20) ensures that a machine cannot process samples while the machine is being idle.

$$Y_{ijn} \leq 1 - Y_{0jn}; \quad \forall\ j \in J,\ i \in I_j, n = 0, \ldots, N \tag{20}$$

Constraints (21)-(28) ensure that the number of samples in a machine is at most equal to the capacity machine j.

$$B_{ijn} \leq \beta_p Y_{ijn}; \quad \forall\ p \in P,\ j \in J_p,\ i \in I_j \cup \{0\},\ n = 0, \ldots, N \tag{21}$$

$$Y_{ijn} \leq B_{ijn}; \quad \forall\ p \in P,\ j \in J_p,\ i \in I_j \cup \{0\},\ n = 1, \ldots, N \tag{22}$$

$$\sum_{i \in I_j \cup \{0\}} B_{ijn} \leq \beta_p Z_{jn}; \quad \forall\ p \in P,\ j \in J_p,\ n = 0, \ldots, N \tag{23}$$

$$BR_{ijn} \leq \beta_p YR_{ijn}; \quad \forall\ p \in P,\ j \in J_p,\ i \in I_j \cup \{0\},\ n = 0, \ldots, N \tag{24}$$

$$\sum_{i \in I_j \cup \{0\}} BR_{ijn} \leq \beta_p (1 - Z_{jn}); \quad \forall\ p \in P,\ j \in J_p,\ n = 0, \ldots, N \tag{25}$$

$$BE_{ijn} \leq \beta_p YE_{ijn}; \quad \forall\ p \in P,\ j \in J_p,\ i \in I_j \cup \{0\},\ n = 0, \ldots, N \tag{26}$$

$$YE_{ijn} \leq BE_{ijn}; \quad \forall\ p \in P,\ j \in J_p,\ i \in I_j \cup \{0\},\ n = 1, \ldots, N \tag{27}$$

$$\sum_{i \in I_j \cup \{0\}} BE_{ijn} \leq \beta_p Z_{jn}; \quad \forall\ p \in P,\ j \in J_p,\ n = 0, \ldots, N \tag{28}$$

Constraint (29) introduces a_{ik} samples for task i at processing unit p_k^i at the beginning of the scheduling horizon as the number of samples waiting to start processing at that unit.

$$W_{ik0} = a_{ik}; \quad \forall i \in I,\ k = 1, \ldots, n(i) \tag{29}$$

Given that the samples are introduced as samples waiting to start processing at the beginning of the scheduling horizon, all machines are set to idle for the first time point according to constraint(30). Note that the optimal schedule can still have machines starting processing

at the beginning of the scheduling horizon if $T_1 = 0$.

$$Y_{0j0} = 1; \quad \forall j \in J \tag{30}$$

Constraint (31) ensures that the number of samples that finishes processing or continues to be processed at machine j at time point n is equal to the number of samples that started or continued to be processed in machine j at the previous time points, i.e., samples are not created or lost.

$$BR_{ijn} + BE_{ijn} = BR_{ij(n-1)} + B_{ij(n-1)}; \quad \forall\ j \in J,\ i \in I_j,\ n = 1, \ldots, N \tag{31}$$

Constraint (32) ensures that a sample from task i can visit processing unit p_k^i in S_i at time point n, if it has already visited processing unit p_{k-1}^i in S_i before time point n.

$$\sum_{j \in J_p:\ p = p_k^i} B_{ijn} + W_{ikn} = W_{ik(n-1)} + \sum_{j \in J_p:\ p = p_{k-1}^i} BE_{ijn}; \quad \forall\ i \in I, \quad k = 2, \ldots, n(i),\ n = 1, \ldots, N \tag{32}$$

Constraint (33) is a flow constraint similar to Constraint (32), but for the first processing unit of path S_i.

$$\sum_{j \in J_p:\ p = p_1^i} B_{ijn} + W_{i1n} = W_{i1(n-1)}; \quad \forall\ i \in I,\ n = 1, \ldots, N \tag{33}$$

Constraints (34)-(35) regulate the amount of time remaining to finish processing samples from task i at machine j at time point n if machine j is set to continue processing samples from task i at time point n.

$$TR_{ijn} \leq \tau(p) Y R_{ijn}; \quad \forall\ p \in P,\ j \in J_p,\ i \in I_j \cup \{0\},\ n = 0, \ldots, N \tag{34}$$

$$TR_{ij(n+1)} \geq TR_{ijn} + \tau(p) Y_{ijn} - SL_{n+1}; \quad \forall\ p \in P,\ j \in J_p,\ i \in I_j,\ n = 0, \ldots, N-1 \tag{35}$$

Note that constraint (34) is not considered for the idle task, since a machine can stop being idle at any time if a set of samples are assigned to start being processed at the machine at that time.

The objective function for the continuous-time formulation is expressed so that it can be directly comparable to the objective function for the flexible discrete-time formulation presented in (8).

$$maximize \sum_{n=0}^{N} \sum_{i=1}^{|I|} \sum_{k=1}^{n(i)} \sum_{j \in J_p : p = p_k^i} \frac{k}{n(i)} B_{ijn} \tag{36}$$

3 Computational Study

In this section, we present the case study, composed of a total of 190 instances and 1,030 runs, based on an actual analytical services facility. In Section 3.1, we define the network of processing units defined by various paths, we present model parameters used for this study, we present the metrics by which comparisons were made, and we present the discretization schemes that were compared. In Sections 3.1 – 3.4, we present and discuss the results of these comparisons.

3.1 Experimental data and design of instances

The network of the processing units being considered in this work is presented in Figure 1. This network resembles the actual analytical services facility that was studied. The identity of the industrial partner and the names of involved processes are not disclosed as per our non-disclosure agreement with the industrial partner. However, the proposed computational studies retain the characteristics of the majority of the operations at this facility. This network of 25 processing units is defined by the 11 paths presented in Table 1. Each

task entering the facility goes through processes A, B, C, and D prior to starting processes defined by the paths in Table 1. For example, a task with Path ID, P1, will have a path of A-B-C-D-Z-F-M-P-K; the first part of the path, A-B-C-D, is the default precursor path, and Z-F-M-P-K is the unique path defined in Table 1.

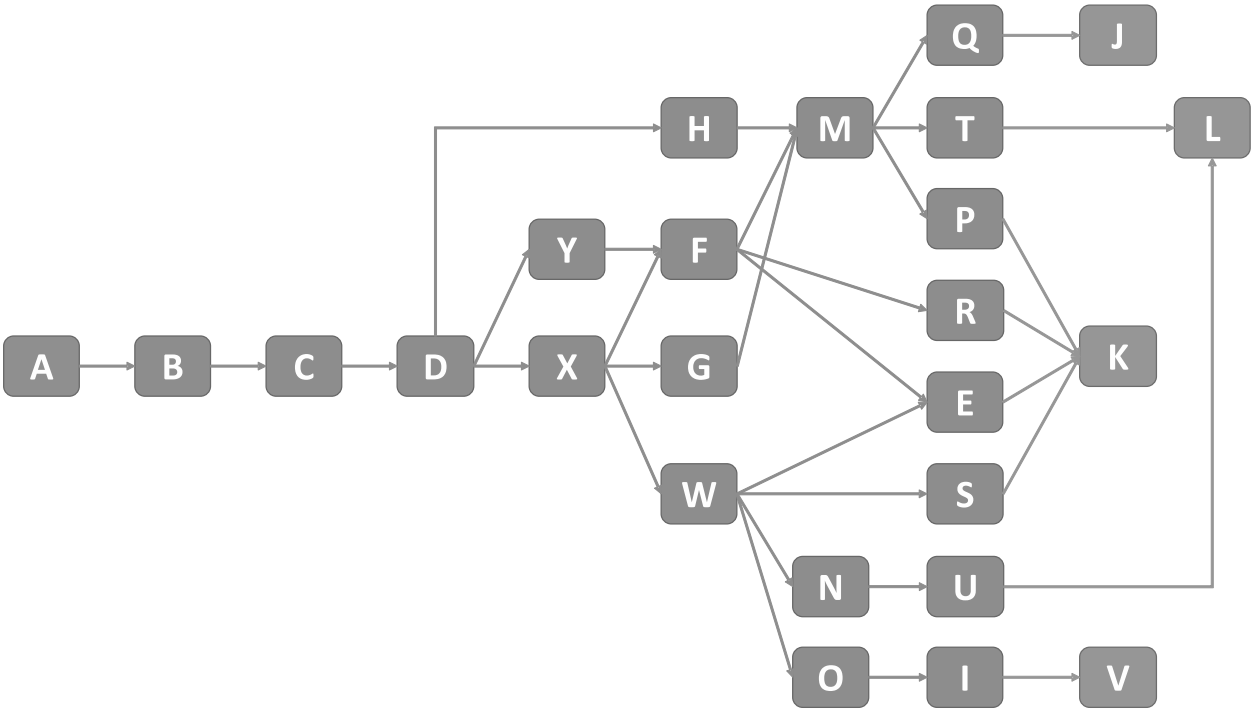

Figure 1: Process map of the network

In this work, the resources in each process unit p have the same processing capacity β_p and processing time $\tau(p)$. The number of resources, the capacity, and the processing time for each processing unit considered for this study are reported in Table 2. Instances created with the process information values reported in Table 2 will be referred to as baseline instances.

In all computational studies, experiments are designed with the following experimental

Table 1: Paths of analytical processes defining the process network. Default path precursor A-B-C-D considered in the analysis but omitted in the table for brevity.

Path ID	Process 5	Process 6	Process 7	Process 8	Process 9
P1	X	F	M	P	K
P2	X	F	E	K	
P3	X	G	M	T	L
P4	X	F	R	K	
P5	X	F	M	T	L
P6	Y	F	M	Q	J
P7	H	M	T	L	
P8	X	W	S	K	
P9	X	W	N	U	L
P10	X	W	O	I	V
P11	X	W	E	K	

Table 2: Process capacity, resources, and processing time information

Process	Capacity	Resources	Processing Time (min)
A	500	2	15
B	60	3	60
C	2250	2	1440
D	50	4	375
E	42	2	40
F	216	2	300
G	21	2	150
H	48	1	615
I	7	1	1440
J	150	1	240
K	480	3	180
L	440	2	240
M	216	4	120
N	440	4	220
O	1	1	10
P	180	1	390
Q	240	1	1440
R	720	10	735
S	480	10	471
T	112	8	1256
U	135	8	1141
V	22	1	60
W	440	4	1620
X	10	6	10
Y	10	1	10

design parameters: number of tasks ($|I|$), length of the scheduling horizon (H), number of time points for the continuous-time formulation (N), and processing time variability (Γ), which has been defined in this work as follows::

$$\Gamma = \frac{\max\limits_{p \in P} \tau(p) - \min\limits_{p \in P} \tau(p)}{\min\limits_{p \in P} \tau(p)} \tag{37}$$

Note that in the context of these computational studies, processing time variability as defined by (37) does not refer to variability of processing time with respect to a dependent variable, but it is a relative measure of the discrepancy between the longest and the shortest processing times considered in these instances. The baseline instances have processing time variability of 161.

In the present work, a set of instances contains ten instances with the same experimental design parameters. However, for each of the ten instances within a set, each task considers:
a) a randomly generated number of samples between 10 and 500, which is the typically observed range of task size at the analytical services facility under study,
b) a randomly assigned Path ID and the path associated with it, and
c) a randomly generated starting point within the path.

In what follows, *UD* will refer to implementation of the discrete-time formulation with uniform time discretization, and *NUD* will refer to implementation of the discrete-time formulation with non-uniform time discretization. When UD or NUD is followed by a number, the number will indicate the discretization scheme. For UD, the number will indicate the size of the uniform time steps. For example, each $p \in P$ in UD10 will have a time step $\Delta(p) = 10$. For NUD, the number will indicate the maximum discretization value. For example, for NUD30, each $p' \in P$ with $\tau(p') < 30$ will have a time step $\Delta(p') = \tau(p')$ and each $p'' \in P$ with $\tau(p'') \geq 30$ will have a time step $\Delta(p'') = 30$. For either UD or NUD, given

a time step $\Delta(p)$ for processing unit p, the the unit-specific time grid is discretized according to (38).

$$\mathcal{E}(p) = (0, 0, \Delta(p), 2\Delta(p), \ldots, (\lceil \frac{H}{\Delta(p)} \rceil - 2)\Delta(p), (\lceil \frac{H}{\Delta(p)} \rceil - 1)\Delta(p), H) \tag{38}$$

$\mathcal{E}(p)$ has $(\lceil \frac{H}{\Delta(p)} \rceil - 1) + 3$ time points index from $t = 0$ to $t = |\mathcal{E}(p)| - 1$. Note that if $\Delta(p)$ is not a factor of H, then $H - \varphi_{p(|\mathcal{E}(p)|-2)} \neq \Delta(p)$. For this reason, Equation (1) defined the time steps up to $t = |\mathcal{E}(p)| - 2$ rather than $t = |\mathcal{E}(p)| - 1$.

Based on the above, UD10, UD30, UD60, NUD30, and NUD60 discretization schemes are studied in the following computational studies. In practice, for short-term scheduling, we would like to be able to make decisions at least on an hourly basis, but it can also becomes impractical to define schedules down to the minute. We consider UD10 to be the finest discretization where the uniform $\Delta(p)$ equals to the minimum allowable accuracy, whereas UD60 represents the *coarse* discretization where the uniform $\Delta(p)$ equals to the maximum allowable accuracy. For all sets of instances, each instance was solved for all five of these discretization schemes, and the continuous-time formulation was solved in addition to these five discretization schemes when the continuous-time formulation was compared against the flexible discrete-time formulation.

Regarding the continuous-time formulation, each of the instances was solved for a selected number of time points, and instances with different numbers of time points were considered to analyze the effect of increasing number of time points on the computational time. Increasing the number of time points, starting with a small number of time points, will increase both the solution quality and computational demand, until a point when adding time points will not improve the quality of the solution. The procedure of starting with a small number of time points and increasing it until no improvement can is observed was used by Ierapetri-

tou and Floudas[21]. However, before the maximum number of useful time points is reached, the problem may become intractable, or fail to be solved to optimality within a given CPU time limit. In this work, we initialized the continuous-time formulation with 6 time points, and aimed to increase the number of time points until no improvement in the solutions are detected. A CPU time limit of 8 hours was considered for these simulations. This CPU time limit was imposed based on the expected time needed to arrive to optimal solutions for industrial-case problems, especially in a short-term scheduling context for analytical service facilities (i.e., a new schedule may be required every day).

For the remainder of this study, the computation of a particular discretization scheme or the continuous-time formulation has been referred to as a *run*. To summarize, a set of instances has ten instances with the same experimental design parameters, and each instance has five or six instances depending on whether or not the continuous-time formulation is considered for that instance.

In principle, UD60, as the *coarse* discretization scheme, may result in the worst objective value and the lowest computational time among those five discretization schemes, and the continuous-time formulation should also yield better solutions than UD60 perhaps at the cost of additional computational time, provided enough number of time points are specified in the continuous-time formulation. Note that comparing absolute values of the objective function and the computational time between different instances may not provide insightful or meaningful conclusions due to different experimental design parameters. Therefore, to evaluate each approach fairly, we compute, for each run within an instance, the 'Relative Objective Benefit (ROB)' and the 'Relative CPU time Disadvantage (RCD)' compared to the objective value and CPU time yielded by UD60 for that instance. ROB and RCD are defined as follows:

$$ROB(\chi) = \frac{ObjVal(\chi) - ObjVal(UD60)}{ObjVal(UD60)} \quad (39)$$

$$RCD(\chi) = \frac{CPUtime(\chi) - CPUtime(UD60)}{CPUtime(UD60)} \quad (40)$$

where χ is one of the discretization schemes or the continuous-time formulation under consideration.

When the mean of a value over a set of instances is reported, the associated standard error of the mean (SEM) is calculated and reported as follows:

$$SEM = \frac{STDEV.S}{\sqrt{NIST}} \quad (41)$$

where STDEV.S is the sample standard deviation, and NIST is the number of instances.

As stated in the Introduction, the following computational experiments are expected to evaluate the strengths and weaknesses of the different time representation approaches relative to each other, and by doing so, determine the favorable approach to solving industrial-sized short-term scheduling problems for a multipurpose plant. All the computational experiments were performed on a Linux server with 250 GB of RAM and 4 CPUs, each with 12 cores and a processing speed of 2.4 GHz using IBM ILOG CPLEX Optimizer 12.6.0.[34] CPLEX was run with the default settings, except for the CPU time limit of 8 hours. Particularly, this means that the setting for parallelism allowed utilization of all 48 available logical cores.

3.2 Comparing the discrete-time and the continuous-time formulations

In order to compare the different discretization schemes along with the continuous-time formulation, 80 relatively small sized instances (i.e., eight sets of instances, each set with ten randomly generated instances with the same experimental design parameters) were considered with the number of tasks ranging from 5 to 10, the number of time points for the continuous-time formulation ranging from 6 to 8, with a set scheduling horizon of 8 hours. We aimed to start with a low number of time points for the continuous-time formulation, and increase the number of time points until the computational time for the continuous-time formulation becomes prohibitive. For each instance, six runs were made for the five discretization schemes and the continuous-time formulation.

Results from these 480 runs are shown in Figure 2, where each point on the plot represents a run; there are 400 points shown on Figure 2 since 80 of the 480 runs are for UD60, whereas $ROB(UD60) = 0$, $RCD(UD60) = 0$ as per Equations (40) and (41) respectively. Thus, these values are not shown for brevity. This allows us to gain some useful insight regarding the trade-off between the quality of the solution and the associated computational cost for the different schemes. As shown in Figure 2, a perfectly vertical trend would represent improved relative solution quality at absolutely no deterioration in relative computational cost. On the other hand, a perfectly horizontal trend would represent deterioration in relative computational cost with absolutely no relative gain in solution quality. Therefore, a time representation scheme with a steep trend resembling a vertical line close to the vertical axis would be generally more desirable over a scheme with a flat trend resembling a horizontal line far from the vertical axis. The horizontal axis for the RCDs is given in logarithmic scale for readability. Note that with the logarithmic scale, a trend that is closer to the vertical axis has a steeper slope and a narrower spread compared to a trend that appears to have similar spread and slope, but further from the vertical axis.

In addition to the slope of the trend, the positions of the scattered points are important performance indicators of a particular scheme. Having scattered points closer to the vertical axis indicates the ability to provide solutions quickly, while having scattered points far from the horizontal axis indicates the ability to provide high quality solutions. Comparing the maximum ROB values (the heights of the trends) of the different schemes is a simple way to compare the potentials of the different schemes to provide high quality solutions.

The mean ROB and RCD values for each set of ten instances and their associated standard errors are reported in Tables 3–6. All runs reached optimality for the flexible discrete-time formulation, whereas 72 of 80 runs for the continuous-time formulation reached optimality, and 8 runs failed to reach optimality within a CPU time limit of 8 hours. The CPU time limit was set to 8 hours in order to be able to complete these runs within a reasonable time frame. More detailed information including the optimality gaps on these 8 runs are reported in Table A1 in the Appendix.

In Figure 2, we can observe the discrete-time formulation forming steep vertical trends and the continuous-time formulation forming a relatively flat trend. For the multi-tasking modeling framework presented in Section 2 for this case study, the discrete-time formulation was also generally able to provide higher quality solutions than the continuous-time formulation. In particular, NUD30, NUD60, UD10, and UD30 returned significantly higher maximum ROB values compared to the continuous-time formulation. Figure 2 suggests that the modeling framework developed in this work may not be suitable for solving large instances while using the continuous-time formulation as computational cost deteriorates rapidly for a small gain in solution quality.

As reported in Table 3, the continuous-time formulation generally provided better quality

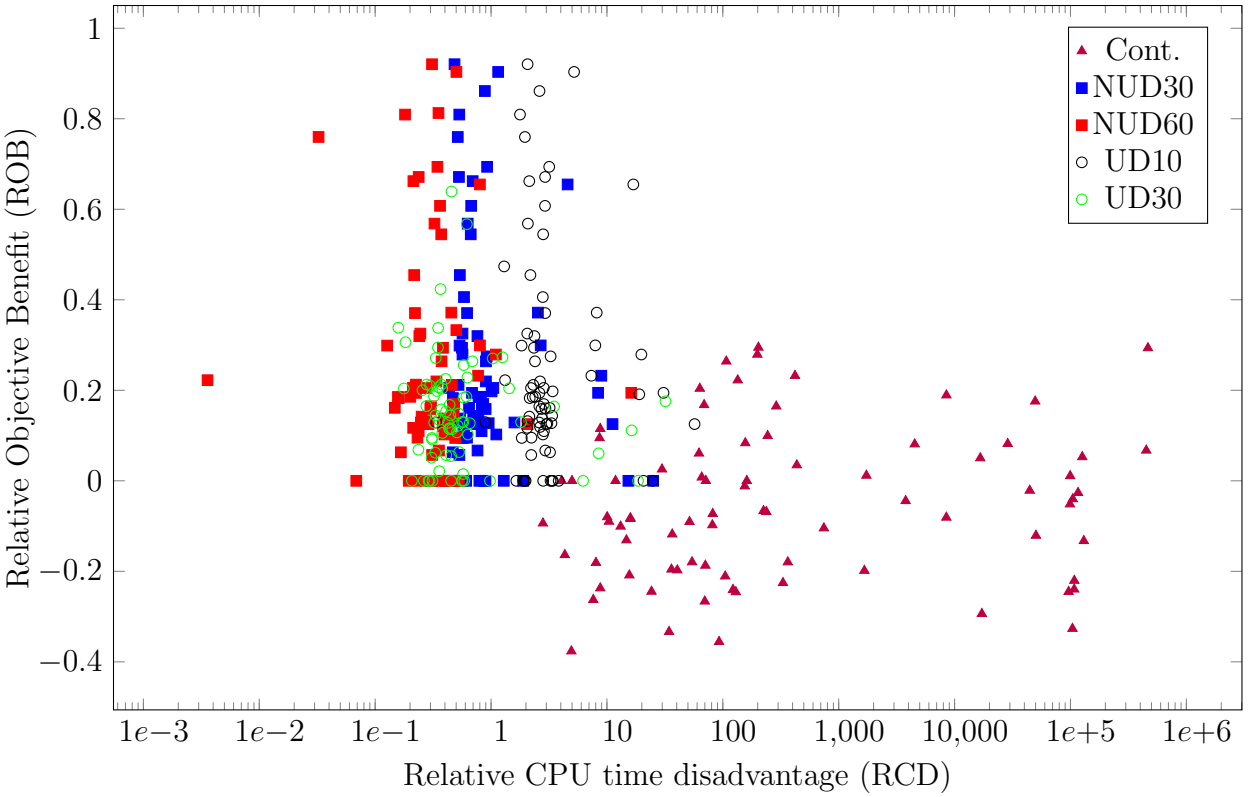

Figure 2: Comparison of different time representations

solutions than UD60 with 5 tasks when implemented with more than 6 time points. However, as reported in Table 5, the computational time increased rapidly for the continuous-time formulation with increasing number of time points, as well as increasing number of tasks. The continuous-time formulation required, on average, 395 seconds for a run with 5 tasks and 6 time points, but a run with 5 tasks and 8 time points required 4,939 seconds, and a run with 10 tasks and 6 time points required 9,153 seconds. On the other hand, NUD30, NUD60, UD10, and UD30 all produced lower mean RCDs with 10 tasks than they did with 5 tasks, generally requiring less than 8 seconds to compute.

These results are in line with the observations of optimality gap at the root node relaxation, the number of variables, and the number of constraints as reported in Tables 7, 8, and 9. While the continuous-time formulation returned lower optimality gap at the root node relaxation than UD10 and UD60 for certain instances, it returned, on average, much higher

optimality gap compared to its NUD and UD counterparts. This indicates that, within the scope of this work, the flexible discrete-time formulation generally led to tighter problems than the continuous-time formulation presented in this work. In addition, the average value of this gap for the continuous-time formulation increased 3 fold when the number of time points was increased from 6 to 8, which indicates a significant trade-off between computational time and solution quality. The continuous-time formulation also produced much higher average numbers of variables and constraints compared to its NUD and UD counterparts, indicating much higher computational demand.

Furthermore, while Standard Error of Mean (SEM) values of ROB were lower for the continuous-time formulation compared to the discrete-time formulation as reported in Table 4, SEM values of RCD were much higher for the continuous-time formulation as reported in Table 6, and this gap in SEM values of RCD increased with increasing number of tasks and increasing number of time points. This indicates that, for the application studied in this work, while the continuous-time formulation may provide solutions with more consistent quality, it may be computationally less consistent, and the CPU time consistency deteriorates with increasing problem size.

Table 3: Mean Relative Objective Benefit (ROB) over UD60

Number of Tasks	Number of Time Points (C)	NUD30	NUD60	UD10	UD30	C	UD60 ObjVal
5	6	0.24	0.23	0.24	0.16	0.00	747
5	7	0.24	0.23	0.24	0.16	0.08	747
5	8	0.24	0.23	0.24	0.16	0.06	747
6	6	0.35	0.35	0.35	0.19	−0.05	898
7	6	0.25	0.25	0.25	0.13	−0.15	1198
8	6	0.22	0.22	0.22	0.14	−0.11	1178
9	6	0.22	0.22	0.22	0.14	−0.14	1205
10	6	0.19	0.18	0.19	0.13	−0.17	1245

These results suggest that the discrete-time representation is more suitable for the present

Table 4: Standard Error of Mean (SEM) of ROB over UD60

Number of Tasks	Number of Time Points (C)	NUD30	NUD60	UD10	UD30	C
5	6	0.04	0.04	0.04	0.03	0.00
5	7	0.04	0.04	0.04	0.03	0.00
5	8	0.04	0.04	0.04	0.03	0.00
6	6	0.11	0.11	0.11	0.05	0.00
7	6	0.05	0.05	0.05	0.02	0.00
8	6	0.07	0.07	0.07	0.04	0.00
9	6	0.07	0.07	0.07	0.03	0.00
10	6	0.06	0.06	0.06	0.02	0.00

Table 5: Mean Relative CPU time disadvantage (RCD) over UD60

Number of Tasks	Number of Time Points (C)	NUD30	NUD 60	UD10	UD30	C	UD60 CPU time (s)
5	6	2.83	0.68	8.01	3.10	506	0.78
5	7	2.83	0.68	8.01	3.10	1444	0.78
5	8	2.83	0.68	8.01	3.10	6331	0.78
6	6	0.61	0.26	2.18	0.37	6912	0.24
7	6	0.73	0.28	2.55	0.37	13 172	0.26
8	6	0.69	0.29	2.63	0.38	17 790	0.27
9	6	0.77	0.34	2.91	0.47	26 828	0.27
10	6	0.79	0.33	3.17	0.49	33 898	0.27

Table 6: Standard Error of Mean (SEM) of RCD over UD60

Number of Tasks	Number of Time Points (C)	NUD30	NUD60	UD10	UD30	C
5	6	1.04	0.55	2.31	1.31	1.40
5	7	1.04	0.55	2.31	1.31	3.03
5	8	1.04	0.55	2.31	1.31	17.13
6	6	0.05	0.04	0.09	0.04	15.53
7	6	0.05	0.03	0.10	0.02	40.26
8	6	0.03	0.03	0.17	0.04	42.02
9	6	0.09	0.04	0.12	0.05	51.88
10	6	0.05	0.03	0.07	0.04	54.28

Table 7: Mean optimality gap (%) at the root node relaxation

Number of Tasks	Number of Time Points (C)	NUD30	NUD60	UD10	UD30	UD60	C
5	6	1.16	0.76	6.38	3.25	0.49	93.33
5	7	1.16	0.76	6.38	3.25	0.49	73.65
5	8	1.16	0.76	6.38	3.25	0.49	288.43
6	6	0.00	0.55	7.46	5.25	0.81	54.69
7	6	0.97	0.26	2.68	1.75	0.16	56.75
8	6	1.51	0.00	6.17	0.00	16.67	52.99
9	6	5.11	0.00	6.30	1.33	0.00	76.58
10	6	0.87	0.00	8.35	2.11	0.00	148.98

Table 8: Mean number of variables for the discrete and continuous time formulations

Number of Tasks	Number of Time Points (C)	NUD30	NUD60	UD10	UD30	UD60	C
5	6	2669	1968	5671	2042	1135	23 569
5	7	2669	1968	5671	2042	1135	27 608
5	8	2669	1968	5671	2042	1135	31 705
6	6	3078	2270	6541	2355	1309	27 276
7	6	3477	2568	7391	2661	1479	31 059
8	6	3930	2894	8351	3007	1671	34 780
9	6	4331	3206	9171	3302	1835	38 598
10	6	4687	3455	9991	3597	1999	42 345

Table 9: Mean number of constraints for the discrete and continuous time formulations

Number of Tasks	Number of Time Points (C)	NUD30	NUD60	UD10	UD30	UD60	C
5	6	2280	1661	4903	1793	1015	21 034
5	7	2280	1661	4903	1793	1015	24 759
5	8	2280	1661	4903	1793	1015	28 968
6	6	2493	1821	5347	1958	1111	23 636
7	6	2702	1978	5780	2120	1204	26 851
8	6	2937	2151	6270	2302	1310	29 483
9	6	3146	2315	6688	2458	1400	32 998
10	6	3332	2448	7106	2614	1490	35 923

multi-tasking modeling framework since it is able to handle industrial-sized short-term scheduling problems of this nature with hundreds of tasks. However, the continuous-time formulation demonstrated that it can obtain higher quality solutions than the coarse uniform discretization scheme when dealing with small numbers of tasks. One advantage of the continuous-time formulation over the coarse uniform discretization scheme is that while the coarse uniform discretization scheme can only provide schedules with events occurring at coarsely predefined time points (e.g., every hour) events can occur anywhere along the scheduling horizon for a schedule provided by the continuous-time formulation. For some scheduling problems involving small number of tasks, for example, scheduling for an industrial chemical plant producing small number of products but in large volumes with long processing times should be able to successfully implement a continuous-time formulation, especially if large CPU time limits can be accepted. For these problems, having precise time points can be important, and defining very fine time points over a long scheduling horizon can make the problems intractable. As reported in Table 5, UD10 required more time to compute by several factors compared to the coarser discretization schemes even when handling a small number of tasks. In addition, the continuous-time approaches must be considered if variable machine processing times is a significant feature of the scheduling problem since discrete-time approaches cannot readily handle this feature without resulting in large, computationally demanding formulations.[20,32]

Comparing among the different discretization schemes, NUD30, NUD60, and UD10 provided solutions of very similar quality as shown in Figure 2 and Table 3. UD30 provided significantly worse solutions with little benefit to the computational time, if at all. While UD10 required more time to compute than the other discretization schemes, it still only required seconds to compute. Therefore, when considering short-term scheduling problems with such small number of tasks, implementing a fine uniform discretization scheme may seem to be suitable over coarser discretization schemes as the fine uniform discretization

scheme can provide schedules with more precise event points.

In the following subsections, much larger instances with hundreds of tasks will be considered as the objective of this work is to study industrial-sized problems. However, these large instances are intractable for the continuous-time formulation considered in this study as made evident above with just 6 to 10 tasks, therefore the following subsections will not consider the continuous-time formulation.

3.3 Comparing uniform and non-uniform discretization: varying $|I|$ and H

In order to compare the performances of the different discretization schemes, 90 instances with the number of tasks ranging from 100 to 200 and the length of the scheduling horizon ranging from 24 to 40 hours were considered. The instances with 100 tasks consisted of 25,600 samples on average. Given that the facility under consideration typically receives around 1000 – 4000 samples for every 8 hours of operation, the sizes of these instances reflect operations of an actual analytical services facility with particularly heavy client demands. All 450 runs for these instances reached optimality.

In Figure 3, the different discretization schemes can be distinguished easily. While NUD30, NUD60, and UD10 all provided significantly better solutions than UD60, the computational time deteriorated relatively quickly for UD10 compared to the non-uniform discretization schemes. While the trend for NUD60 is much steeper compared to the trend for UD10 (recall that the horizontal axis is in logarithmic scale), the maximum ROB value of NUD60 on this plot is almost as high as that of UD10. That is, NUD60 provided solutions that were virtually as good as those provided by UD10 at a fraction of the computational cost compared to UD10 (99% less RCD, on average, as reported on Table 12).

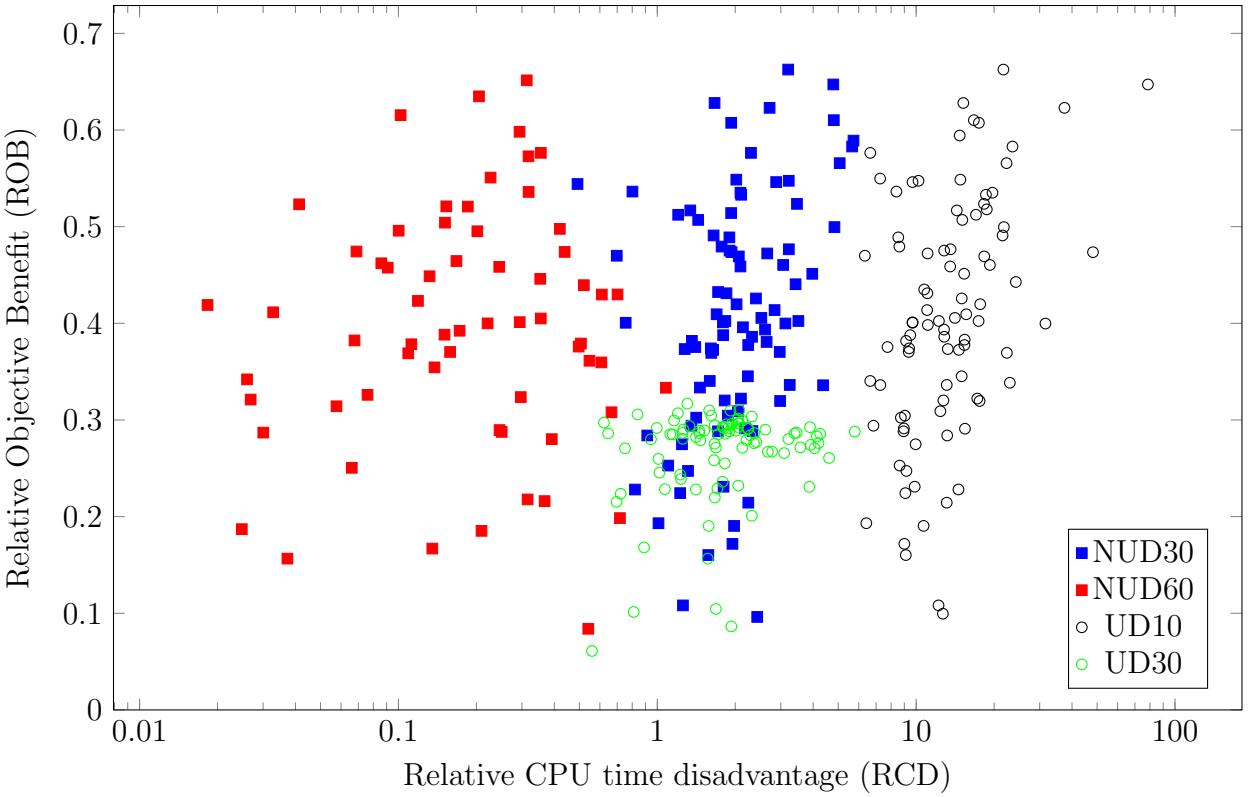

Figure 3: Comparison of different discretization schemes

Superior tractability of NUD60 compared to UD10 is clearly demonstrated in Tables 14–16, which show that NUD60 required, on average, 67% less variables, 66% less constraints, and produced 89% lower optimality gap at the root node relaxation compared to UD10. Similarly, NUD30 also produced, on average, 84% less RCD, required 54% less variables and constraints, and produced, on average, 53% lower optimality gap at the root node relaxation. However, NUD30 provided the same quality of solutions as UD10. While UD30 required around the same amount of time to solve as NUD30, and NUD30 required 34% less variables and 33% less constraints, UD30 produced, on average, 51% and 48% lower ROB compared to NUD30 and NUD60, respectively, as reported on Table 10. Furthermore, while NUD60 required merely 3% less variables and constraints compared to UD30, NUD60 produced, on average, 92% lower RCD, since NUD60 led to much tighter problems with 85% lower optimality gap at the root node relaxation. These results clearly demonstrate the superiority of non-uniform discretization schemes over uniform discretization schemes when

handling large problems of this nature.

Table 10: Mean Relative Objective Benefit (ROB) over UD60

Number of Tasks	Horizon (h)	NUD30	NUD60	UD10	UD30	UD60 ObjVal
100	24	0.28	0.28	0.28	0.22	10 021
100	32	0.29	0.28	0.29	0.24	10 189
100	40	0.29	0.29	0.29	0.25	10 160
150	24	0.40	0.39	0.40	0.28	10 705
150	32	0.46	0.45	0.46	0.29	10 539
150	40	0.45	0.44	0.45	0.29	10 660
200	24	0.47	0.47	0.47	0.27	11 080
200	32	0.55	0.54	0.55	0.28	10 957
200	40	0.45	0.44	0.45	0.27	11 122

Table 11: Standard Error of Mean (SEM) of ROB over UD60

Number of Tasks	Horizon (h)	NUD30	NUD60	UD10	UD30
100	24	0.03	0.03	0.03	0.03
100	32	0.04	0.04	0.04	0.02
100	40	0.03	0.03	0.03	0.02
150	24	0.03	0.03	0.03	0.01
150	32	0.03	0.03	0.03	0.01
150	40	0.02	0.02	0.02	0.00
200	24	0.03	0.03	0.03	0.01
200	32	0.03	0.02	0.03	0.00
200	40	0.03	0.03	0.03	0.01

By inspecting Table 10, we can also observe the relative advantage of finer discretization schemes (NUD30, NUD60, and UD10) becoming more significant compared to the coarser discretization schemes (UD30 and UD60) as higher numbers of tasks are considered. However, by inspecting Table 12, we can observe mean RCD values increasing noticeably with increasing number of tasks for UD10, while the increases in mean RCD values are not significant for NUD30, and the mean RCD values appear to decrease for NUD60 with increasing number of tasks. In addition, while Standard Error of Mean (SEM) values of ROB between NUD30, NUD60, and UD10 were very close as reported in 11, SEM values of RCD were much

higher for UD10, as well as for UD30 to a certain degree, compared to NUD30 and NUD60 as reported in 13, and this gap in SEM values of RCD increased with increasing number of tasks. These results indicate that the gap between non-uniform discretization schemes and uniform discretization schemes only enlarges with increasing problem size, and that the non-uniform discretization schemes perform more consistently with regards to the CPU time.

Table 12: Mean Relative CPU time disadvantage (RCD) over UD60

Number of Tasks	Horizon (h)	NUD30	NUD60	UD10	UD30	UD60 CPU time (s)
100	24	1.90	0.19	10.30	1.77	7.32
100	32	2.22	0.40	12.41	2.21	6.34
100	40	1.47	0.12	9.84	1.85	8.09
150	24	2.26	0.06	14.06	2.04	11.48
150	32	1.97	0.15	13.04	1.59	11.89
150	40	2.06	0.04	14.20	2.14	11.99
200	24	3.18	0.16	28.63	2.47	14.26
200	32	2.91	0.18	16.71	1.97	14.95
200	40	2.17	0.08	14.73	1.85	16.68

Table 13: Standard Error of Mean (SEM) of RCD over UD60

Number of Tasks	Horizon (h)	NUD30	NUD60	UD10	UD30
100	24	0.20	0.06	0.77	0.36
100	32	0.31	0.12	1.57	0.40
100	40	0.12	0.06	0.69	0.26
150	24	0.26	0.06	0.79	0.22
150	32	0.28	0.08	1.05	0.24
150	40	0.36	0.05	1.95	0.42
200	24	0.49	0.08	6.56	0.34
200	32	0.43	0.09	2.68	0.30
200	40	0.28	0.07	1.15	0.35

Table 14: Mean optimality gap (%) at the root node relaxation

Number of Tasks	Horizon (h)	NUD30	NUD60	UD10	UD30	UD60
100	24	1.39	0.43	4.24	1.35	3.13
100	32	1.61	0.34	2.99	2.82	2.36
100	40	3.45	0.73	5.74	7.20	4.92
150	24	2.30	0.03	3.44	3.70	2.78
150	32	0.63	0.42	6.64	1.80	1.48
150	40	1.86	0.13	4.35	4.29	1.49
200	24	0.10	0.46	1.26	1.24	0.85
200	32	1.61	0.20	1.46	3.31	1.65
200	40	2.28	0.12	6.06	1.25	1.40

Table 15: Mean number of variables for the NUD and UD discretization schemes

Number of Tasks	Horizon (h)	NUD30	NUD60	UD10	UD30	UD60
100	24	116 866	84 933	255 501	87 501	45 501
100	32	118 271	86 267	257 165	88 071	45 797
100	40	117 108	84 977	256 377	87 801	45 657
150	24	174 885	127 150	382 112	130 861	68 048
150	32	175 747	127 959	382 930	131 141	68 194
150	40	175 564	127 694	383 134	131 211	68 230
200	24	232 277	168 636	508 256	174 061	90 512
200	32	232 670	169 290	507 818	173 911	90 434
200	40	233 297	169 570	509 366	174 441	90 710

Table 16: Mean number of constraints for the NUD and UD discretization schemes

Number of Tasks	Horizon (h)	NUD30	NUD60	UD10	UD30	UD60
100	24	61 829	45 054	134 525	46 637	24 665
100	32	62 537	45 727	135 362	46 927	24 818
100	40	61 953	45 079	134 966	46 790	24 746
150	24	91 272	66 596	198 264	68 750	36 372
150	32	91 706	67 004	198 675	68 893	36 447
150	40	91 615	66 872	198 778	68 929	36 466
200	24	120 400	87 771	261 768	90 782	48 036
200	32	120 595	88 097	261 547	90 706	47 995
200	40	120 914	88 242	262 326	90 976	48 138

3.4 Comparing uniform and non-uniform discretization: effects of processing time variability

The preceding computational studies clearly demonstrated the superiority of non-uniform discretization schemes compared to uniform discretization schemes in solving large instances of the nature of problems considered in this study. However, those instances only considered instances with the baseline processing time variability Γ of 161. Note that at Γ of 1, NUD30 and NUD60 would be equivalent to UD10, where 10 minutes is the smallest given processing time. Therefore, as Γ approaches 1 from the baseline value of 161, the relative performance superiority of NUD30 and NUD60 over UD10 as observed previously should gradually diminish. To observe whether or not the conclusions drawn previously for the baseline instances still holds at relatively lower values of Γ still exceeding 1, the set of instances with 100 tasks and 24 hour scheduling horizon as used in Section 3.3 were re-run at Γ values of 40 and 100. To set the Γ value to 40, for example, minimum processing time was kept at 10 minutes, and any processing times exceeding 400 minutes were set to 400 minutes. Reducing Γ down to 40, we can still observe superior efficiency of the non-uniform discrete schemes compared to the uniform discrete schemes, as shown in Figure 4.

Since we are interested in observing how the relative performance superiority of NUD30 and NUD60 over UD10 changes with changing Γ, the mean ROB values at their corresponding Γ values, as well as their SEM values given as error bars, are presented in Figure 5 instead. The mean ROB values at each Γ value were spaced out horizontally for readability. Note that in Figure 5, the mean ROB values for NUD30, NUD60, and UD10 appear close to each other for $\Gamma = 161$. For $\Gamma = 40$, there is clearly a gap between NUD60 and the finer discretizations, NUD30 and UD10. This indicates that the superiority of non-uniform discretization schemes becomes less pronounced with lower processing time variability. When large CPU time limits can be accepted, some of the finer uniform discretization schemes may be more appealing than some of the coarser non-uniform discretization schemes, especially

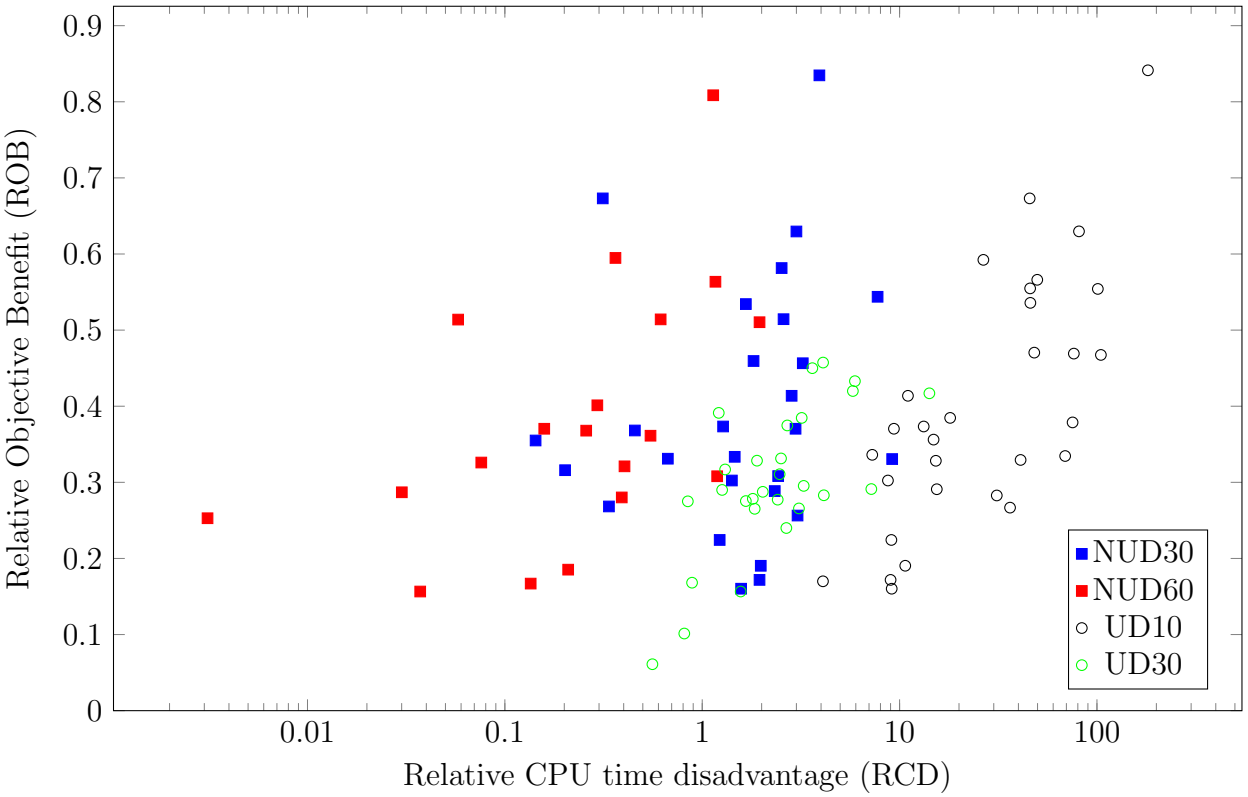

Figure 4: NUD vs UD; changing processing time variability Γ

for instances with relatively lower numbers of tasks given the observations made in Section 3.3. Similarly, for problems with low processing time variability, finer non-uniform discretization schemes may be more desirable than coarser non-uniform discretization schemes.

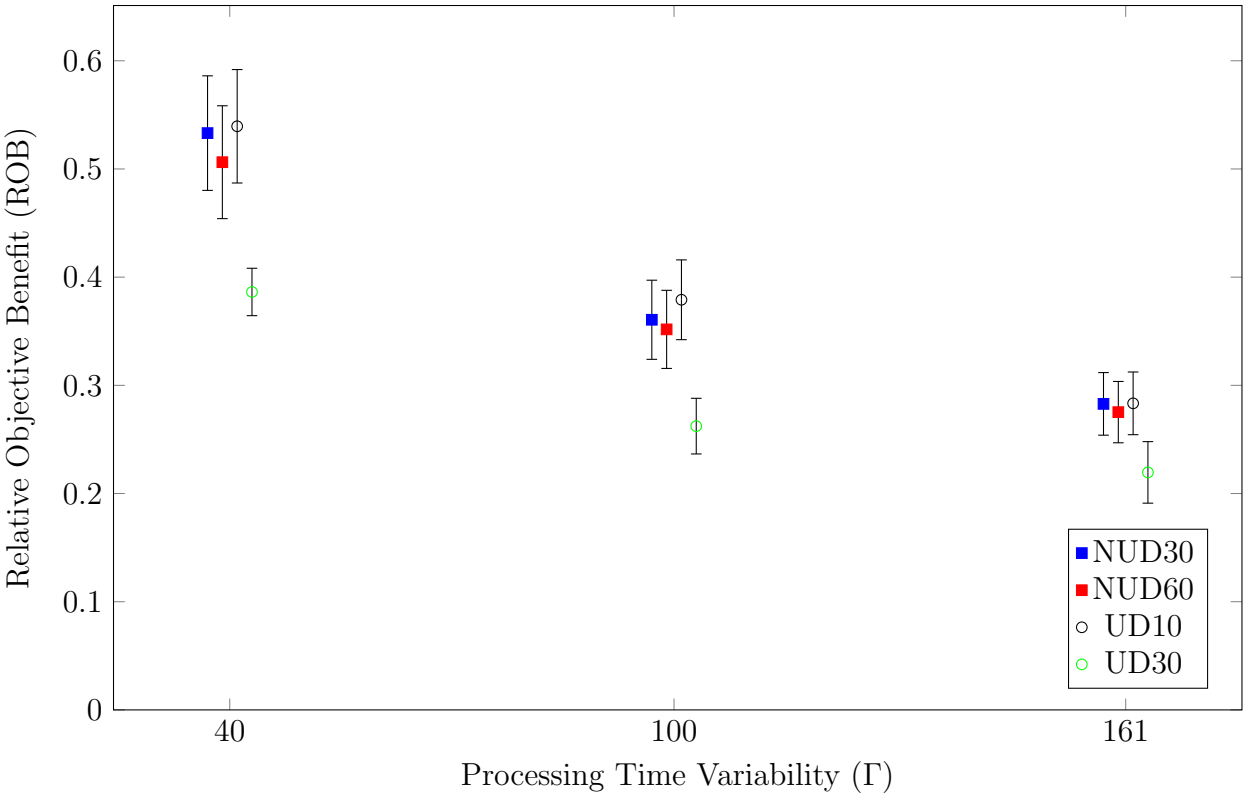

Figure 5: Trend of change in ROB with changing processing time variability Γ

4 Conclusion

This study presents a comparison between a flexible discrete-time formulation, with both uniform and non-uniform time discretizations, and a continuous-time formulation in the context of short-term scheduling of operations in an analytical services facility. Through a series of computational experiments designed based on an actual analytical services facility, the strengths and weaknesses of the different time representation approaches relative to each other were studied in order to determine the favorable approach to solving industrial-sized short-term scheduling problems for a multipurpose plant.

Based on the results of our computational study, the multitasking continuous-time formulation based on conservation of flow proposed by Lagzi et al.[20] may not be suitable for addressing large-scale problems. For the instances considered in this work, the flexible

discrete-time formulation led to tighter and smaller problems compared to the continuous-time formulation. Computational cost for the continuous-time formulation became prohibitive, taking hundreds to tens of thousands of times longer to compute compared to the flexible discrete-time formulation, while almost all of the instances resulted in poorer solution quality for the continuous-time formulation. The performance of the continuous-time formulation presented in this work deteriorates even more with increasing problem size. These results are in contrast to some of the previous results reported in the literature on comparing the performance of discrete (without flexible time discretization capability) and continuous-time formulations.[31] This contrast highlights the importance of further investigations to be conducted on discrete-time formulations, since the conclusions may vary from application to application.

For the flexible discrete-time formulation, non-uniform discretization schemes solved large instances more efficiently compared to the uniform discretization schemes within the scope of this study. While the quality of solutions for the non-uniform discretization schemes was virtually as good as the quality of solutions for the finest uniform discretization scheme, the computational times for the non-uniform discretization schemes were merely fractions of the computational times for the finest uniform discretization scheme, as the non-uniform discretization schemes generally led to tighter problems with smaller numbers of variables and constraints. The superior relative performance of the non-uniform discretization schemes become even more apparent with increasing problem size, as the computational cost deteriorates much more rapidly for the uniform discretization schemes compared to the non-uniform discretization schemes. Superior relative performance of the non-uniform discretization schemes also become more pronounced with increasing processing time variability. For problems with low processing time variability, however, a fine uniform discretization scheme might be more desirable compared to a relatively coarse non-uniform discretization scheme if solution quality is of utmost importance while long CPU time limits can be tolerated.

In terms of future research, there are multiple directions to take. For discrete-time approaches, one direction is to explore algorithms for discretizing the multiple and non-uniform time grids for the flexible discrete-time approaches, which do not necessarily produce time grids with equally spaced time points. These advanced discretization schemes may provide better balance between precision of event points and the computational cost.

For the continuous-time approaches, one direction is to explore alternatives that would improve the solution time for the continuous-time approaches by developing formulations with smaller number binary variables, using new model representations (e.g., state-task network, STN), or by tightening the LP relaxations of the existing formulations. Unit-specific continuous-time formulations may also be extended to include more operational characteristics, such as multitasking. Researchers have been successful in reducing formulation size and computational demands by implementing unit-specific approaches.[21–24,27,28] Development of a unit-specific continuous-time formulation accounting for multitasking, while significantly reducing formulation size and computational demands, is considered outside the scope of the present study and would be considered as part of the future research work.

5 Acknowledgment

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

6 Appendix

The runs reported in Table A1 are the runs that failed to reach optimality within eight hours. These runs still appear in Figure 2 and are included in mean value calculations in Section 3.2. Here, optimality gap is calculated as:

Table A1: Runs that failed to reach optimality within eight hours

Run Type	Number of Tasks	Number of Samples	Horizon (hr)	Number of Time Points	Optimality Gap (%)
Continuous	6	1624	8	6	1.99
Continuous	7	1275	8	6	5.36
Continuous	8	2975	8	6	16.01
Continuous	9	2163	8	6	5.41
Continuous	9	2845	8	6	10.96
Continuous	9	2555	8	6	13.37
Continuous	10	1803	8	6	16.39
Continuous	10	2367	8	6	13.55
Continuous	10	2742	8	6	9.92
Continuous	5	1418	8	7	0.44
Continuous	5	1440	8	8	3.9
Continuous	5	1364	8	8	2.13

$$\frac{UB - OBJ}{UB} \tag{42}$$

where UB denotes the smallest upper bound on the objective function value obtained from the branch-and-bound tree search, and OBJ is the objective function value of the best feasible solution found within the CPU time limit of eight hours.

References

(1) Kondili, E.; Pantelides, C. C.; Sargent, R. W. H. A General Algorithm for Short-Term Scheduling of Batch-Operations . I. Milp Formulation. *Comput. Chem. Eng.* **1993**, *17*, 211–227.

(2) Shah, N.; Pantelides, C. C.; Sargent, R. W. H. A General Algorithm for Short-Term

Scheduling of Batch-Operations. II. Computational Issues. *Comput. Chem. Eng.* **1993**, *17*, 211–227.

(3) Pantelides, C. C. Unified frameworks for optimal process planning and scheduling. Proceedings on the second conference on foundations of computer aided operations. **1994**, pp 253–274.

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

(5) Velez, S.; Maravelias, C. T. Multiple and nonuniform time grids in discrete-time MIP models for chemical production scheduling. *Comput. Chem. Eng.* **2013**, *53*, 70–85.

(6) Bassett, M. H.; Penky, J.; Reklaitis, G. V. Using Detailed Scheduling To Obtain Realistic Operating Policies for a Batch Processing Facility. *Ind. Eng. Chem. Res.* **1997**, *36*, 1717–1726.

(7) Velez, S.; Maravelias, C. T. Theoretical framework for formulating MIP scheduling models with multiple and non-uniform discrete-time grids. *Comput. Chem. Eng.* **2015**, *72*, 233–254.

(8) Velez, S.; Merchan, A. F.; Maravelias, C. T. On the solution of large-scale mixed integer programming scheduling models. *Chem. Eng. Sci.* **2015**, *136*, 139–157.

(9) Zhang, X.; Sargent, R. W. H. The optimal operation of mixed production facilitiesa general formulation and some approaches for the solution. *Comput. Chem. Eng.* **1996**, *20*, 897–904.

(10) Schilling, G.; Pantelides, C. A simple continuous-time process scheduling formulation and a novel solution algorithm. *Comput. Chem. Eng.* **1996**, *20*, S1221–S1226.

(11) Mcdonald, C. M.; Karimi, I. A. Planning and Scheduling of Parallel Semicontinuous Processes . 1 . Production Planning. *Society* **1997**, *36*, 2691–2700.

(12) Mockus, L.; Reklaitis, G. V. Continuous time representation approach to batch and continuous process scheduling. 2. Computational issues. *Ind. Eng. Chem. Res.* **1999**, *38*, 204–210.

(13) Castro, P.; Barbosa-Póvoa, A.; Matos, H. An Improved RTN Continuous-Time Formulation for the Short-term Scheduling of Multipurpose Batch Plants. *Ind. Eng. Chem. Res.* **2001**, *40*, 2059–2068.

(14) Lee, K. H.; Park, H. I.; Lee, I. B. A novel nonuniform discrete time formulation for short-term scheduling of batch and continuous processes. *Ind. Eng. Chem. Res.* **2001**, *40*, 4902.

(15) Wang, S.; Guignard, M. Redefining event variables for efficient modeling of continuous-time batch processing. *Ann. Oper. Res.* **2002**, *116*, 113–126.

(16) Maravelias, C. T.; Grossmann, I. E. New General Continuous-Time StateTask Network Formulation for Short-Term Scheduling of Multipurpose Batch Plants. *Ind. Eng. Chem. Res.* **2003**, *42*, 3056–3074.

(17) Gupta, S.; Karimi, I. A. An Improved MILP Formulation for Scheduling Multiproduct, Multistage Batch Plants. *Ind. Eng. Chem. Res.* **2003**, *42*, 2365–2380.

(18) Sundaramoorthy, A.; Karimi, I. A. A simpler better slot-based continuous-time formulation for short-term scheduling in multipurpose batch plants. *Chem. Eng. Sci.* **2005**, *60*, 2679–2702.

(19) Li, J.; Xiao, X.; Tang, Q.; Floudas, C. A. Production scheduling of a large-scale steelmaking continuous casting process via unit-specific event-based continuous-time mod-

els: Short-term and medium-term scheduling. *Ind. Eng. Chem. Res.* **2012**, *51*, 7300–7319.

(20) Lagzi, S.; Fukasawa, R.; Ricardez-Sandoval, L. A multitasking continuous time formulation for short-term scheduling of operations in multipurpose plants. *Comput. Chem. Eng.* **2017**, *97*, 135–146.

(21) Ierapetritou, M. G.; Floudas, C. A. Effective Continuous-Time Formulation for Short-Term Scheduling. 1. Multipurpose Batch Processes. *Ind. Eng. Chem. Res.* **1998**, *37*, 4341–4359.

(22) Lin, X.; Floudas, C. A. Design, synthesis and scheduling of multipurpose batch plants via an effective continuous-time formulation. *Comput. Chem. Eng.* **2001**, *25*, 665–674.

(23) Giannelos, N. F.; Georgiadis, M. C. A Simple New Continuous-Time Formulation for Short-Term Scheduling of Multipurpose Batch Processes. *Ind. Eng. Chem. Res.* **2002**, *41*, 2178–2184.

(24) Janak, S. L.; Lin, X.; Floudas, C. a. Enhanced Continuous-Time Unit-Specific Event-Based Formulation for Short-Term Scheduling of Multipurpose Batch Processes: Resource Constraints and Mixed Storage Policies. *Ind. Eng. Chem. Res.* **2004**, *43*, 2516.

(25) Castro, P. M.; Grossmann, I. E.; Novais, A. Q. Two new continuous-time models for the scheduling of multistage batch plants with sequence dependent changeovers. *Ind. Eng. Chem. Res.* **2006**, *45*, 6210–6226.

(26) Furman, K. C.; Jia, Z.; Ierapetritou, M. G. A robust event-based continuous time formulation for tank transfer scheduling. *Ind. Eng. Chem. Res.* **2007**, *46*, 9126–9136.

(27) Mouret, S.; Grossmann, I. E.; Pestiaux, P. A novel priority-slot based continuous-time formulation for crude-oil scheduling problems. *Ind. Eng. Chem. Res.* **2009**, *48*, 8515–8528.

(28) Susarla, N.; Li, J.; Karimi, I. A. A novel approach to scheduling multipurpose batch plants using unit-slots. *AIChE J.* **2010**, *56*, 1859–1879.

(29) Lappas, N. H.; Gounaris, C. E. Multi-stage adjustable robust optimization for process scheduling under uncertainty. *AIChE J.* **2016**, *62*, 1646–1667.

(30) Wang, Z.; Li, Z.; Feng, Y.; Rong, G. Crude-Oil Operations under Uncertainty: A Continuous-Time Rescheduling Framework and a Simulation Environment for Validation. *Ind. Eng. Chem. Res.* **2016**, *55*, 11383–11401.

(31) Stefansson, H.; Sigmarsdottir, S.; Jensson, P.; Shah, N. Discrete and continuous time representations and mathematical models for large production scheduling problems: A case study from the pharmaceutical industry. *Eur. J. Oper. Res.* **2011**, *215*, 383–392.

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

(33) Merchan, A. F.; Lee, H.; Maravelias, C. T. Discrete-time mixed-integer programming models and solution methods for production scheduling in multistage facilities. *Comput. Chem. Eng.* **2016**, *94*, 387–410.

(34) IBM ILOG, CPLEX Optimization Studio. **2013**; `http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud`.

For Table of Contents (TOC) Only