Proceedings of the 2023 Winter Simulation Conference
C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.

AN APPROACH TOWARDS PREDICTING THE COMPUTATIONAL RUNTIME REDUCTION FROM DISCRETE-EVENT SIMULATION MODEL SIMPLIFICATION OPERATIONS

Mohd Shoaib
Varun Ramamohan

Department of Mechanical Engineering
Indian Institute of Technology Delhi
New Delhi-110016
INDIA

Navonil Mustafee

Centre for Simulation, Analytics and Modelling
University of Exeter
Rennes Drive, Exeter, EX4 4ST
UNITED KINGDOM

ABSTRACT

Model simplification is the process of developing a simplified version of an existing discrete-event simulation (DES) to study the performance of specific system subcomponents relevant to the analysis. The simplified model is referred to as a 'metasimulation'. A widely used model simplification operation is abstraction, which involves replacing the subcomponents, not core to the analysis, from the parent DES model with random variables representing the lengths of stay in said subcomponents. However, the one-time computational cost of developing metasimulations via abstraction can itself be considerable, as the approach necessitates executing the parent model for generating the necessary data for developing the metasimulation. Thus, this study proposes a queuing-theoretic approach for estimating the computational runtime reduction (CRR) achieved through abstraction, wherein the prediction of CRR precedes the development of the metasimulation. Towards this, we present preliminary results from applying this approach for simplification of DES models made up of $M/M/n$ workstations.

1 INTRODUCTION

A simulation originally developed to model a large and complex system may need to be adapted for conducting experiments requiring only a subset of the model. In such cases, a simplified version of the original simulation may reduce the computational overheads associated with the analysis. For instance, Adan et al. (2014) developed a simplified version of a larger hospital simulation to determine appropriate capacity levels for the emergency department. The simplification was achieved by omitting patient treatment details from the main model. The process of developing simplified versions of larger simulation models is known as 'model simplification'. We henceforth refer to the original 'full-featured' simulation model as the *parent* simulation and the associated simplified model as the *metasimulation* (analogous to 'metamodel').

Model simplification reduces the computational expense associated with model execution (Johnson et al. 2005; Piplani and Puah 2004). It is a complexity reduction exercise in which the components and connections in the parent models are modified so as to decrease computational expense with little or no change in overall model behavior, including retention of the overall stochastic nature of simulation outputs (Brooks and Tobias 2000). In the extant literature, several studies have reported a non-trivial reduction in model execution times brought about through simplified models. For example, Fatma et al. (2020) claimed a runtime reduction of up to 80%, whereas Hung and Leachman (1999) reported a reduction of approximately 20%. In addition to reduced model runtime, simplified inputs and a shortened development cycle have been documented as some of the potential benefits of model simplification (Brooks and Tobias 2000; van der Zee 2017).

Model simplification operations primarily include *aggregation*, *substitution* or *abstraction*, and *deletion* of model components. These can be used in isolation or in combination with other operations (Rank et al.

2016; Fatma et al. 2020). Note that the terms abstraction and substitution are used interchangeably in the literature; however, we use the term abstraction to be consistent with recently published work (Fatma et al. 2020). Abstraction involves reducing the size and complexity of the model by substituting the redundant subsystems or components from the detailed model with constant-valued delays (Johnson et al. 2005; Brooks and Tobias 2000) or random variables (Fatma et al. 2020) representing the length of stay in the subsystem. Consequently, in the simplified model, only the subsystems crucial to the modeling aim are retained in their full-featured form. An illustration of abstraction using a two-stage tandem queuing system and its simplified version is shown in Figure 1. It consists of two different subsystems, each comprising one server and one queue. The figure depicts the abstraction of the second subsystem (enclosed in the dotted rectangle in the parent model in Figure 1a) with a random variable representing the length of stay in subsystem 2 (Figure 1b). The objective of the simplification exercise is to reduce the computational costs associated with the execution of the parent model while still retaining a full-featured representation of subsystem 1 (most relevant to the analysis at hand) as well as the total length of time spent in the system.

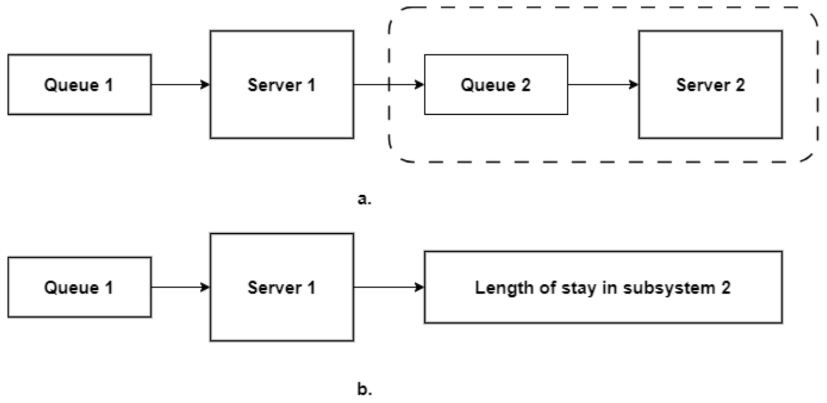

Figure 1: a. A two-stage tandem queuing system (parent model). b. A metasimulation (child) model of the parent model developed by substituting the second subsystem with a random variable representing the length of stay in the second subsystem.

On the other hand, in aggregation, multiple process-flows or components/products are combined to form a single unified compound process-flow or machine/product (Lidberg et al. 2021). An example of aggregation can be found in Fatma et al. (2020), in which six beds in the inpatient department of a hospital were aggregated into a single bed, and the service rates were adjusted accordingly to retain the overall utilization of the beds.

The model simplification process itself can be computationally expensive, with a one-time computational cost incurred as part of metasimulation model development and validation. Therefore, access to a sufficiently accurate prediction of the expected computational runtime reduction (CRR) at the conceptualization stage of model simplification could be crucial for modelers to decide whether to pursue model simplification at all and, if yes, what level of simplification they can consider. The question of whether the extent of CRR can be determined before the simplified model has been constructed and executed has not yet been addressed in the literature. To this end, we propose a queuing-theoretic based approach to predict the computational savings associated with the metasimulation model at the point of its conceptualization before executing the parent model for generating the data required to develop the metasimulation. This approach can potentially be applied to general DES models developed to model service system operations, given that they can typically be considered as complex queuing systems made up of simpler queuing subsystems, especially if the subsystems have FCFS queuing disciplines. However, given that this is a first step towards generating *a priori* CRR predictions associated with model simplification operations, the scope of the paper is presently confined to computational implementation for DES models made up of M/M workstations.

2 LITERATURE SURVEY

According to Zeigler et al. (2000), model complexity can be defined in terms of the number of elements, connections, and model calculations. Further, the author proposed that the complexity of a model can be reduced by replacing certain elements of the detailed model with a random variable (abstraction), combining components of a model (aggregation), or restricting the set of values that the variable can assume.

Over time, numerous model simplification strategies have been developed, including *deletion*, *aggregation*, and *abstraction* (Rank et al. 2016; Pegden et al. 1995; Lidberg et al. 2021). However, *abstraction* and *aggregation* are the two most widely adopted techniques, with the common objective of reducing the computational overhead without compromising the probability distributions of key simulation outcomes. As discussed in the subsequent section, the abstraction operation is the focus of this study and, thus, is the subject of the literature review.

Abstraction, *model condensation* or *substitution* (Piplani and Puah 2004) can be described as the process of reducing the non-essential components from a detailed model. The common practice, as observed in the literature, is to replace specific components or elements from the detailed model with either constant delays (Hung and Leachman 1999; Johnson et al. 2005) or random-valued delays (Fatma et al. 2020; Etman et al. 2011).

Rose (1998) used a simple simulation model to characterize the behavior of a wafer fabrication facility. The author demonstrated the effectiveness of simplified simulations by studying the evolution of the work-in-process after a catastrophic bottleneck failure. The model consisted of a full-featured bottleneck and an abstraction—represented by an aggregated delay—of the remaining machines in the process flow. In a subsequent study, Rose (1999) presented a statistical analysis of the lot interarrival time at the bottleneck workstation. The probability distribution best describing the behavior of the data was identified and used to model the arrivals at the bottleneck machine. In a follow-up study, Rose (2007) expanded the application of the simple model by introducing inventory-dependent distributions for sampling the delay instead of the fixed distributions used in earlier studies.

Further, for a semi-conductor manufacturing unit, tool set substitution along with process flow aggregation was conducted to reduce the model complexity (Stogniy and Scholl 2019; Stogniy and Scholl 2020). The abstraction of the toolset was done using delays representing the time for loading, unloading, processing, and waiting. Similarly, Hung and Leachman (1999) replaced low utilization workstations with constant and also random valued variables representing the lengths of stay corresponding to waiting time and processing time. Völker and Gmilkowsky (2003) included only the relevant processes in the simplified model and substituted the others using the time lags corresponding to the processing and the waiting times. Additionally, Fatma et al. (2020) developed simplified simulation models of a network of primary care facilities in healthcare settings. The authors considered only the components relevant to their analysis and replaced the non-core elements with random variables representing the patient lengths of stay (wait time + service time) at each element.

Our literature review has shown that despite the many benefits of model simplification, including quicker development cycles, additional flexibility, and faster execution, its use by researchers has only been sporadic (van der Zee 2017). Of the studies that exist, the primary focus has been on the simplification of models to reduce their overall complexity and computational expense without compromising their utility and credibility. Typically, models are simplified before runtime reductions are realized. In contrast, in this study, we propose estimating the expected computational runtime reduction (CRR) at the conceptualization stage of metasimulation development, after which model simplification may be undertaken if deemed necessary. To the best of our knowledge, this is the first study to propose an approach whereby CRR estimates are generated before the metasimulation is developed. Note that our study assumes that the representational capability of the metasimulations with respect to the parent model is adequate, and hence we do not discuss their validation in detail, and focus on generating accurate CRR predictions.

3 COMPUTATIONAL RUNTIME REDUCTION PREDICTION METHODOLOGY

We first investigate the mechanism by which CRR is achieved by the abstraction model simplification operation. This knowledge of the CRR mechanism helps motivate the key insights underpinning the development of CRR prediction models.

3.1 Determinants of the CRR Achieved with Metasimulations

The DES models - both parent and metasimulation models - that are considered in this study are all programmed in the Python programming platform using the *salabim* DES library (van der Ham 2018). We now describe the terminology used in the development of our CRR prediction models. First, the term 'event list' refers to an ordered record of the simulation's pending events prepared so that they are executed in the correct order. On the other hand, 'simulation trace', or simply 'trace', can be defined as the time-ordered record of already elapsed events associated with the execution of a simulation. It is typically available as standard output in most simulation platforms, including *salabim.* Next, the 'generator' can broadly be defined as the function that generates objects or simulation entities arriving at the system seeking service. The term 'instructions' is used in this paper to refer to messages that are relayed or passed by the processor to execute the events from the event list. For instance, standard instructions may include: '*generate component*', '*enter queue*', '*leave queue*', and '*release server*'.

We now briefly discuss the determinants of simulation runtime for DESs, which influence the choice of independent variables for the CRR prediction models. The simulation execution time can be considered to be a function of the number of simulation subsystems, the utilization of the servers of each subsystem, and the simulation run length. Therefore, for a fixed simulation run length, the number of subsystems and the server utilization primarily determine the model execution time. An increase in either of these values will lead to a corresponding rise in the number of instructions or events on the event list. However, there is a distinction between the nature of the increase in the number of instructions due to the increase in the number of subsystems and the number of arrivals that merits discussion. As the number of subsystems increases, keeping other factors fixed, the number of instructions per arrival does not change, but the total number of instructions increases. On the contrary, the number of instructions per arrival changes (which in turn determines the total number of instructions) based on the utilization of the servers in the system. The server utilization determines the probability of a new arrival finding an idle server for receiving service. If this probability is low, the service-seeking entity will likely receive service as soon as it arrives, implying that fewer instructions are required. If this probability is high, more instructions are required to coordinate the arrival of the entity, its service request, its entry into the queue, its activation when it is due for service, and so on. In the context of *salabim*, when server utilization is high, the instruction set may include, for example, *passivate entity, activate entity* and *request on hold.* When server utilization is low, the system may not need to execute the instructions to passivate and activate the service-seeking entity given that it is highly likely to receive service immediately upon arrival, and thus the number of instructions per unit arrival would be significantly lower.

For a given model simplification task via the abstraction operation, the number of subsystems - either in the parent model or in the metasimulation - do not change, and hence we consider only the utilization of the subsystems that make up the parent model subcomponent to be abstracted out as a determinant of the number of instructions per arrival. A brief note regarding notation: we use the symbol θ to denote the number of instructions per arrival and the symbol I for the total number of instructions associated with a given DES system and its execution.

We begin the development of the CRR prediction models by examining the list of instructions obtained from the trace of a simulation. For illustration, the trace output from the two-stage $M/M/1$ queuing system in Figure 1a (referred to henceforth as system $2s$) and its corresponding metasimulation model in Figure 1b (system *ms* henceforth) was extracted and compiled. Table 1 provides the retrieved trace output highlighting the instructions executed by the processor for an arrival from generation to exit from both models. In the table, *queue 1* and *resource 1* denote the queue and the server in both the first stage of the parent model and in the metasimulation. *arrival.1* refers to the first entity created by the generator. Observe that the metasimulation trace lacks *resource 2* and *queue 2*, which correspond to the parent model's second subsystem.

Table 1: Sample instructions from a two-stage $M/M/1$ tandem queuing system and corresponding metasimulation model with second stage abstracted out.

Parent model	**Metasimulation**
Component generator: 1. Current 2. Hold for $\sim exp\left(\frac{1}{\lambda}\right)$ time	Component generator: 1. Current 2. Hold for $\sim exp\left(\frac{1}{\lambda}\right)$ time
Component process: 1. Create 2. Activate 3. Current 4. Enter *queue*1 5. Request *resource*1 6. Request for *resource*1 is scheduled for time $=\infty$, goes to passive state[a] 7. Request honored, scheduled for time $=t$ 8. At time $=t$, current (from passive state) 9. Leave *queue*1 10. Claims *resource 1* and holds for $\sim exp\left(\frac{1}{\mu}\right)$ 11. At time $t_1 = t + \sim exp\left(\frac{1}{\mu}\right)$, current 12. Release *resource*1 13. Enter *queue*2 14. Request *resource*2 15. Request for the *resource*2 - scheduled for time$=\infty$, goes to passive state[a] 16. Request honored, scheduled for time $=t_2$ 17. At time $=t_2$, current 18. Claims resource and holds for $\sim exp\left(\frac{1}{\mu'}\right)$ 19. At time $t_3 = t_2 + \sim exp\left(\frac{1}{\mu'}\right)$, current 20. Release *resource*2 21. End	Component process: 1. Create 2. Activate 3. Current 4. Enter *queue*1 5. Request *resource*1 6. Request for the *resource*1 is scheduled for time $=\infty$, goes to passive state[a] 7. Request honored, scheduled for time $=t$ 8. At time $=t$, current (from passive state) 9. Leave *queue*1 10. Claims *resource 1* and holds for $\sim exp\left(\frac{1}{\mu}\right)$ 11. At time $t_1 = t + \sim exp\left(\frac{1}{\mu}\right)$, current 12. Release *resource*1 13. Schedule for time $= t_1 + \sim t_*$[b] 14. At time $= t_1 + \sim t_*$, current 15. End

Notes. $\sim$ indicates 'sample from distribution'. λ = mean arrival rate. μ, μ' = mean service rates of resources 1 and 2, respectively. t_* = length of stay in subsystem 2. [a]Resource occupied, component waits for its turn. [b]Random variable representing length of stay in subsystem 2 (abstracted out).

It is evident that there were fewer instructions per arrival for the metasimulation model. This reduction in the number of instructions is most likely responsible for the decrease in the model runtime. This observation led us to develop a model for predicting CRR from the abstraction model simplification operation as a function of the expected Reduction in Number of Instructions (RNI) with the metasimulation, which in turn we modeled as a function of server utilization.

We briefly discuss impact of two other aspects of DES systems on computational runtime needs to be addressed: (a) the number of servers at a queueing subsystem and (b) the number of process routings in a model on the computation time. An $M/M/n$ queuing model was developed to determine how the number of servers (resources) affects the model runtime. The experiment involved varying the number of servers from 1000 to 100 in decrements of 100 and further decreasing the number of servers to 25 in decrements of 25. The service rate was adjusted proportionally for each case such the server utilization remained unaffected. The computational runtime in minutes was then recorded for each configuration, with a given system configuration determined by the number of servers. This experiment, in essence, mimics the 'aggregation' model simplification operation.

We observe from this experiment that the computational runtime decreases only when the number of servers reaches 100, after which it remains approximately constant. This is likely because, especially at moderate utilization levels (for example, approximately 50%), as the number of servers increases, more computational effort is expended in searching for the first idle server for the job assignment. For this study, we assume that the number of servers in the workstations within the systems that are modeled is much fewer than 100, and hence we no longer consider the number of servers in a queuing workstation/subsystem as a contributing factor to the CRR that can be achieved by abstraction. This is also why we do not consider the aggregation model simplification operation involving replacing multi-server workstations with single-server workstations in this study.

We also do not consider process routings, the logical paths that simulation entities follow within the model, as a factor affecting computational runtimes. This is because unless the number of routings is very high, implying a very high degree of complexity of the simulation model, it is unlikely that the number of instructions will be significantly affected.

From the above discussion, the insight driving the development of the CRR prediction approach emerges as follows. The reduction in the total number of instructions associated with the metasimulation (when compared to the parent model) drives the reduction of its computational runtime, and, for a fixed simulation run length, the number of instructions associated with either the parent simulation or the metasimulation is a function of the utilization of the subsystems being abstracted out. This implies that the key model required for predicting the CRR associated with the metasimulation is a relationship between the RNI achieved when the abstraction model simplification operation is performed and the corresponding CRR. However, developing this model will, in turn, require estimating the RNI achieved with the metasimulation.

3.2 CRR Prediction Model Development Process

At this stage, we note that the CRR prediction needs to be generated prior to the development of the metasimulation model, and ideally without the need to execute the parent simulation for the aforementioned purpose. However, it is assumed that a table of key outcomes from the parent simulation, including utilization estimates for each subsystem being abstracted, is available. Therefore, the CRR prediction must ideally be generated as a function of one of these available outcomes. We recall here that the utilization of the abstracted subsystem is the key determinant of the number of instructions associated with it. Thus we first develop a model that captures the relationship between the number of instructions per arrival and its utilization. The output of this model is used to derive the total number of instructions associated with the subsystem being abstracted. From this, the number of instructions associated with the corresponding representation of this subsystem in the metasimulation is subtracted, yielding the RNI achieved by the metasimulation, and can then be provided as input to the model that relates CRR to the RNI.

We now discuss the fact that developing these models will involve regressing data for the CRR and the RNI. This data can be generated by considering any DES system and its metasimulation, as the independent variable is only the RNI, and not the number of instructions itself. Thus, we consider the parent model and its metasimulation depicted in Figure 1 - i.e., the systems $2s$ and ms for the development of this relationship. Secondly, the DES system ss, consisting of a queue and a server with exponentially distributed service times (without a generator), can be considered the fundamental building block of most DES systems. For example, the $2s$ system can be considered to be made up of a generator and two ss systems in tandem. This is also keeping with the fact that M/M workstations with multiple servers were found not to have any more of an impact on computational runtime than a single-server M/M system as long as the number of servers was approximately less than 100. Thus, when abstracting out a subcomponent of a DES, the subcomponent will be made up of these ss systems. Hence the RNI associated with the abstracted out subcomponent in the metasimulation can be calculated as a function of the sum of the number of instructions associated with each *ss* system (in turn calculated as a function of its utilization) making up the subcomponent. It is therefore important to develop a relationship between the number of instructions associated with an *ss* system and its utilization. This is why we consider the 2*s*, *ms* and *ss* systems in developing the CRR prediction approach.

The CRR prediction model development process can be divided into two stages. The first stage involves the development of models to predict the number of instructions per arrival from the parent and metasimulation models. The second stage uses the models developed in the first stage to derive the CRR prediction model.

The overall approach is described below.

Stage 1. Model: Number of instructions per arrival versus server utilization for each DES system $k \in K = \{2s, ms, ss\}$.
I. For each $k \in K$ and for multiple utilization values of the subsystem being abstracted out, perform the following steps. Note that for the 2*s* system, the subsystem in question is the second stage queue and server, as depicted in Figure 1, and for the *ms* system, this refers to the first stage queue and server. In constructing the CRR prediction models, we assume that the utilization of both servers in the 2*s* system is the same.

(a) Execute DES for k^{th} system and obtain trace as output.
(b) Use the trace as input to record the average number of instructions per arrival, θ_k.

II. Using data collected from step 1.I, for each $k \in K$, fit regression model between θ_k (dependent variable) and server utilization (independent variable).
Stage 2. Model: CRR versus RNI achieved with metasimulation.
I. Perform the below steps for $k \in \{2s, ms\}$ systems for multiple values of server utilization.

(a) Record the average execution time (t^k) and the average number of arrivals (n_k).
(b) Using the regression model obtained from Stage 1 relating θ_k and server utilization, estimate the expected number of instructions per arrival (θ_k).
(c) Estimate the average total number of instructions ($I_k = n_k \times \theta_k$).

II. For each value of I_k and t_k $(k \in \{2s, ms\})$ collected in 2.I, perform the following:

(a) Calculate the RNI as: $\bar{I} = I_{2s} - I_{ms}$.
(b) Calculate the CRR as: $\phi = t_{2s} - t_{ms}$.

III. Using the average CRR values (ϕ) as the dependent variable and the average RNI ($\bar{I}$) as the independent variable, fit a regression model to obtain ϕ as a function of $\bar{I}$.

Note that in Stage 1 we derive a model for predicting the number of instructions per arrival θ as a function of the utilization of the subsystem being abstracted, but then use θ in conjunction with n to estimate the total number of instructions I associated with the execution of the simulation in question to estimate the CRR ϕ (and do not use θ directly). This is because the expression for ϕ estimates the expected CRR in absolute time units (e.g., seconds), and hence the simulation run length, and consequently the number of arrivals during the run length of the simulation, needs to be accounted for when estimating the CRR.

An example of the computational implementation of the above CRR prediction model development process is provided via pseudocode for Stage 2 in Algorithm 1.

Algorithm 1: Pseudocode for the implementation of Stage 2 of the CRR prediction model development process.

```
1  Specify a set of server utilization values $S$, average service rate $\mu$, number of replications ($r$) to be run for each
     server utilization value
2  Determine average arrival rates, $A = \{\lambda : \lambda = s \times \mu, s \in S\}$
3  Initialize arrays (time_list, arrival_list, $\theta^k$_list, $I^k$_list) for data collection
4  for $k \in \{2s, ms\}$ do
5      for $s \in S$ do
6          $i \leftarrow r$
7          while $i \neq 0$ do
8              Run simulation at arrival rate $\lambda$ & server utilization $\mu$
9              Record the number of arrivals ($n_s^{i,k}$) and model runtime ($t_s^{i,k}$) (i.e., time_list.append($t_s^{i,k}$),
                 arrival_list.append($n_s^{i,k}$))
10             $i = i - 1$
11         Obtain the average number of arrivals ($n_s^k$) and average model runtime ($t_s^k$) over $r$ replications, store $t_s^k$ in
             $t^k$_list (e.g., $t_s^k = \frac{sum(time\_list)}{r}$)
12         Estimate $\theta_s^k$ as a function of $s$ using the regression model from Stage 1, store in $\theta^k$_list
13         Calculate $I_s^k = \theta_s^k \times n_s^k$, store in $I^k$_list
14 Calculate reduction in number of instructions $\bar{I} = I^{2s}$_list $-$ $I^{ms}$_list
15 Calculate CRR $\phi = t^{2s}$_list $-$ $t^{ms}$_list
16 Regress $\phi$ on $\bar{I}$
```

4 COMPUTATIONAL IMPLEMENTATION OF THE CRR PREDICTION APPROACH

In this section, we describe the computational experiments conducted to demonstrate the development of the CRR prediction models and the validation of this approach. All computational analyses were conducted on a Windows 11 64-bit workstation with an Intel *i*7 4-core processor with 3.3 GHz clock speed and 16 gigabytes of memory.

4.1 CRR Prediction Model Development: Computational Demonstration

Stage 1 demonstration: development of the θ versus server utilization regression model. The first step in this process was the development of the DES systems $2s$, ms and ss. Mean service times of 1 minute were assumed for each relevant server in each system. In order to gather sufficient data points for the regression, the server utilization levels were varied by 1% between 25% and 92% (beyond 92% nonlinearity in queue outcomes such as average wait time becomes prominent). The server utilization was used to compute the mean interarrival time for service-seeking entities in conjunction with the mean service time.

Having determined the DES input parameters for each simulated system, the parent simulation model (the $2s$ DES) was executed to obtain the trace as the output. The trace was fed as input to a method that returned the number of instructions for every simulation entity listed in the trace. Following this, a predetermined number of entries (50 in our implementation) were randomly selected from each replication to determine the number of instructions per arrival for that replication. Then, 5 such replications were performed at each server utilization value, and correspondingly the average number of instructions per arrival was computed for each value of server utilization. A linear regression model was then fitted to the recorded average number of instructions per arrival and server utilization data. The resulting linear model had an R-squared value of 0.99, indicating a good fit. The same approach was executed for the other systems - the ms and the ss systems to derive expressions for θ_{ss} and θ_{ms} as functions of server utilization. Below, we briefly discuss the construction and validation of the metasimulation (i.e., the ms DES).

Given that the *ms* DES is constructed from the parent 2*s* DES, the second stage in the 2*s* DES was replaced with a random variable representing the length of stay in the second stage. The length of stay values in the second stage obtained from the parent model underwent a distribution-fitting procedure, and the chi-squared distribution was identified to be the best-fitting probability distribution with a *p*-value of 0.99 for the Kolmogorov-Smirnov test. The *ms* DES was then constructed with the second stage in the 2*s* DES replaced (abstracted) by the chi-squared random variable. The *ms* DES was validated against its 2*s* parent DES by conducting a Kullback-Leibler (KL) divergence test comparing the distributions of the total lengths of stay in both systems. The KL test results indicated negligible deviations between the distributions of the length of stay samples from both the 2*s* and the *ms* models.

The linear regression relations relating θ to server utilization obtained for the $2s, ms$ and *ss* systems along with their *R*-squared values are provided in Table 2.

Stage 2 demonstration: the CRR versus RNI regression model. The CRR versus RNI regression model was developed by implementing Algorithm 1. This involved executing the 2*s* DES and its simplified counterpart (the *ms* DES) and recording the total number of arrivals (denoted by *n*) and the model runtime. This process was repeated for every utilization value considered for the experiments - that is, server utilization values between the range 25-92%, in increments of 1%. Following this, the equations for θ_{2s} and θ_{ms} developed via Stage 1 and provided in Table 2 were used to estimate the number of instructions per arrival for each server utilization value. Then the total number of instructions for both models was calculated by multiplying the number of instructions per arrival and the number of arrivals (*n*), after which the difference in the total number of instructions and the computational runtimes between the parent and child model was calculated. A regression line was then fitted to the data for the RNI and the CRR (ϕ). This CRR prediction model is shown in Table 2 along with its *R*-squared value.

Table 2: Linear regression equations comprising the CRR prediction model for the abstraction model simplification operation.

DES	Expression	R^2 value
Two-stage	$\theta_{2s} = 22.02 + 1.97x$	0.99
Single server	$\theta_{ss} = 9.02 + 0.98x$	0.92
Metasimulation	$\theta_{ms} = 15.01 + 0.97x$	0.92
CRR prediction model	$\phi = -0.05 + 0.04\bar{I}$	0.99

x is server utilization, $\bar{I}$ is the RNI with the metasimulation in units of 10000 seconds, and ϕ is the expected CRR.

Note that the average total number of arrivals (*n*) used for calculating the total number of instructions were obtained from the simulation output. However, the value could also be estimated by multiplying the average number of arrivals (estimated from the server utilization and mean service rate) and the total simulation run duration. We developed the CRR prediction model by estimating *n* in this manner as well and the approach yielded comparable results.

We now describe the computational experiments conducted to demonstrate how to apply the CRR prediction approach. Each of these experiments also serve to validate the CRR prediction approach.

4.2 CRR Prediction Model Deployment and Validation

We conducted two sets of experiments as part of the deployment and validation exercise. In the first set, the CRR prediction approach was applied for the 2*s* system DES and its metasimulation represented by the *ms* system DES. The approach for deployment and validation for this case proceeded as follows. For a given server utilization value - we recall here that the 2*s* system was parameterized to have equal utilization values for each server - the number of instructions per arrival θ_{2s} and θ_{ms} were estimated as a function of the server utilization using the relevant equations in Table 2. Then the number of arrivals (n) for a given simulation run length was computed (n = server utilization $\times$ service rate $\times$ simulation run length), and this, in turn, was used to estimate the total number of instructions I_{2s} and I_{ms}. The RNI $\bar{I}$ was calculated as $I_{2s} - I_{ms}$, and this was then used in conjunction with the equation in the last row of Table 2 to predict the CRR ϕ. This process was then repeated for 131 server utilization values between the range 25% to 91%. Note that as part of the above process, neither the $2s$ nor the ms DESs were executed.

For each of the above 131 server utilization values, the actual CRR achieved by the metasimulation was then estimated using the DESs of the parent and metasimulation models. The CRR for each server utilization value was estimated as the average of 30 replicate values and then compared to the predicted values. The performance of the CRR predictions was quantified using two commonly used metrics for regression models: the mean absolute percentage error (MAPE) and the mean percentage error (MPE). The MAPE is calculated as $\sum_{j=1}^{N} \frac{|O_j - P_j|}{O_j} \times 100$, where O_j and P_j are the observed and predicted values of the regression dependent variable, and N represents the size of the sample (131 in this case). Similarly, the MPE is calculated as $\sum_{j=1}^{N} \frac{(O_j - P_j)}{O_j} \times 100$. For this exercise, we observed MAPE and MPE values of 8.61% and 8.15%, respectively. Note that the errors were positively biased, indicating that the prediction model produced conservative CRR estimates.

Further, for the second set of experiments, a more complex system, shown in Figure 2, was created. This parent model is a three-stage tandem system with two parallel systems in the third stage. Two model simplification scenarios were considered as part of this experiment: a) the third stage of the parent model was abstracted out as shown in the second system from the top in Figure 2; and b) the second and third stages were abstracted out together and replaced with a single random variable, depicted in the bottom-most system in Figure 2. Validation similar to that conducted for the $2s$ and ms systems was done for these metasimulations as well. Before we provide the details of the experiments, we first note that both cases involve abstracting out subsystems that are ss systems - i.e., they are $M/M/1$ workstations with a queue and a server, but without a component (i.e., service-seeking entity) generator. Thus in predicting the number of instructions per arrival as a function of the utilization of such a system, the regression equations developed for θ_{ss} in Table 2 are used here. Secondly, we also observe via our computational experiments that abstracting such a system with a random variable representing the length of stay in the system incurs 2 instructions per arrival.

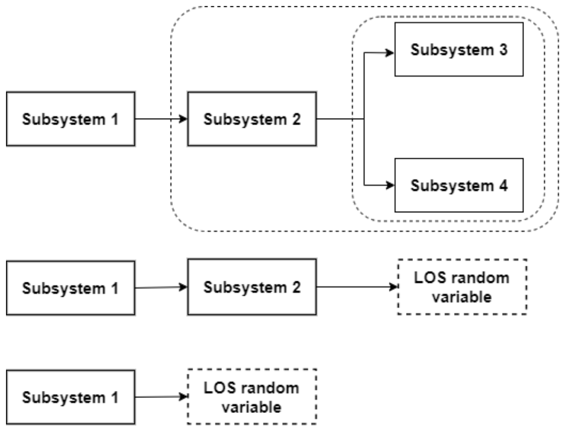

Figure 2: CRR prediction validation: the parent model and two corresponding metasimulation models.

For the first case ten server utilization values for subsystems 3 and 4 were randomly generated between 25% and 92%. First, for subsystems 3 and 4, the expected number of instructions per arrival, which we denote by θ_{ss3} and θ_{ss4}, were computed using the equation for θ_{ss} from Table 2 corresponding to each utilization value. Then, the total number of instructions in both subsystems were obtained by combining the number of arrivals and the number of instructions per arrival. At this point, we note that in calculating the RNI associated with abstracting out the third stage - subsystems 3 and 4 together - will involve summing up the total number of instructions associated with both subsystems and subtracting the number of instructions associated with replacing the third stage by the corresponding length of stay random variable. Thus, the expected RNI $\bar{I}$ associated with this abstraction operation can be calculated as: $\bar{I} = n \times (\theta_{ss3} + \theta_{ss4} - 2)$, where n is the number of arrivals in the desired simulation run duration. The expected CRR is then computed using the regression equation for ϕ from Table 2 as a function of $\bar{I}$.

Next, in order to validate the CRR predictions, the actual CRR associated with this abstraction operation was calculated for each server utilization value in a manner similar to that for the previous validation exercise for the $2s$ and ms systems. The average values of MAPE and MPE observed were 7.37% and 2.89%, respectively.

For the second abstraction scenario, where both stages 2 and 3 are abstracted out by a single random variable, the same approach is followed. In this case, 5 randomly generated server utilization values were considered, and the expected total RNI, using the regression model for θ_{ss} developed earlier, was calculated as follows: $\bar{I} = n \times (\theta_{ss2} + \theta_{ss3} + \theta_{ss4} - 2)$. The expected CRR ϕ was then calculated as a function of $\bar{I}$. The actual CRR values were computed by executing the parent model and its corresponding metasimulation in a manner similar to the first validation exercise, and the observed MAPE and MPE values were 4.81%, and 3.77%.

Overall, given that the MAPE and MPE values for each validation exercise considered above are less than 10%, it is reasonable to conclude that the CRR prediction approach is performing well.

5 DISCUSSION AND CONCLUSION

The paper presents a methodological investigation into DES model simplification operations. We first investigate the mechanism underpinning the CRR achieved through the abstraction model simplification operation. Next, insights from this investigation contributed to the development of a methodology to predict the CRR prior to model simplification. Given that this study is the first step towards being able to generate *a priori* estimates of the expected CRR from model simplification operations, we demonstrate the application of this approach for a relatively simple DES model comprising of M/M queuing subsystems. Based on our computational experiments, the CRR prediction approach performs well.

The prediction approach was developed to determine, at the metasimulation conceptualization stage, whether the model simplification was a worthwhile endeavour, and if so, the extent to which it may be pursued. Such an approach may also serve as the first step towards the incorporation of automated model simplification operations within commercial DES platforms. For example, we envision a scenario where a modeler selects certain subsystems in an existing 'parent' DES for abstraction (e.g., clicks on components in a graphical representation of the modeled system), receives an estimate of the CRR associated with this operation, and if acceptable, then performs the model simplification operation via another 'click' or two.

Future work involves extending this methodology to more general DES models of service systems that can be represented as complex queuing systems. This includes, for example, complex queuing networks made up of G/G systems. Further, incorporating more complex queuing considerations into the approach, such as multi-class service systems, more complex queuing disciplines such as those with reneging and balking, nonstationary arrival and service patterns, etc., will provide a rich set of problems to pursue.

REFERENCES

Adan, I., E. Lefeber, J. Timmermans, A. van de Waarsenburg, and M. Wolleswinkel-Schriek. 2014. "Aggregate Model-based Performance Analysis of an Emergency Department". *International Journal of Privacy and Health Information Management* 2(2):1–21.

Brooks, R. J., and A. M. Tobias. 2000. "Simplification in the Simulation of Manufacturing Systems". *International Journal of Production Research* 38:1009–1027.

Etman, L. F., C. P. Veeger, E. Lefeber, I. J. Adan, and J. E. Rooda. 2011. "Aggregate Modeling of Semiconductor Equipment Using Effective Process Times". In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, 1790–1802. IEEE.

Fatma, N., M. Shoaib, N. Mustafee, and V. Ramamohan. 2020. "Primary Healthcare Delivery Network Simulation Using Stochastic Metamodels". In *Proceedings of the 2020 Winter Simulation Conference*, edited by K. Bae, B. Feng, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, 818–829. IEEE.

Hung, Y. F., and R. C. Leachman. 1999. "Reduced Simulation Models of Wafer Fabrication Facilities". *International Journal of Production Research* 37:2685–2701.

Johnson, R., J. Fowler, and G. Mackulak. 2005. "A Discrete Event Simulation Model Simplification Technique". In *Proceedings of the 2005 Winter Simulation Conference*, 2172 – 2176. IEEE.

Lidberg, S., M. Frantzén, T. Aslam, and A. H. C. Ng. 2021. "Model Simplification Methods for Coded Discrete-Event Simulation Models: A Systematic Review and Experimental Study". http://his.diva-portal.org/smash/record.jsf?pid=diva2%3A1600921&dswid=-1718, accessed 15.12.2022.

Pegden, C. D., R. P. Sadowski, and R. E. Shannon. 1995. *Introduction to Simulation Using SIMAN*. McGraw-Hill, Inc.

Piplani, R., and S. A. Puah. 2004. "Simplification Strategies for Simulation Models of Semiconductor Facilities". *Journal of Manufacturing Technology Management* 15:618–625.

Rank, S., C. Hammel, T. Schmidt, J. Müller, A. Wenzel, R. Lasch, and G. Schneider. 2016. "The Correct Level of Model Complexity in Semiconductor Fab Simulation - Lessons Learned From Practice". In *2016 27th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, 133–139. IEEE.

Rose, O. 1998. "WIP Evolution of a Semiconductor Factory After a Bottleneck Workcenter Breakdown". In *Proceedings of the 1998 Winter Simulation Conference*, edited by D. Medeiros, E. Watson, J. Carson, and M. Manivannan, Volume 2, 997–1003. IEEE.

Rose, O. 1999. "Estimation of the Cycle Time Distribution of a Wafer Fab by a Simple Simulation Model". *Proceedings of the SMOMS* 99(1999):118.

Rose, O. 2007. "Improved Simple Simulation Models for Semiconductor Wafer Factories.". In *Proceedings of the 2007 Winter Simulation Conference*, edited by S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 1708–1712. IEEE.

Stogniy, I., and W. Scholl. 2019. "Using Delays for Process Flow Simplification". In *Proceedings of the 2019 Winter Simulation Conference*, edited by N. Mustafee, K.-H. G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, 2372–2383. IEEE.

Stogniy, I., and W. Scholl. 2020, 12. "Using Accuracy Measurements to Evaluate Simulation Model Simplification". In *Proceedings of the 2020 Winter Simulation Conference*, edited by K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, Volume 2020-December, 1825–1836. IEEE.

van der Ham, R. 2018. "Salabim: Discrete Event Simulation and Animation in Python". *Journal of Open Source Software* 3(27):767.

van der Zee, D.-J. 2017. "Approaches for simulation model simplification". In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, 4197–4208. IEEE.

Völker, S., and P. Gmilkowsky. 2003. "Reduced Discrete-event Simulation Models for Medium-term Production Scheduling". *Systems Analysis Modelling Simulation* 43:867–883.

Zeigler, B. P., T. G. Kim, and H. Praehofer. 2000. *Theory of Modeling and Simulation*. Academic Press.

AUTHOR BIOGRAPHIES

Mohd Shoaib is a Ph.D. candidate at the Department of Mechanical Engineering at the Indian Institute of Technology Delhi. His email address is mohd.shoaib.nitie@gmail.com.

Navonil Mustafee is a Professor at the University of Exeter Business School. He holds a Ph.D. in Information Systems and Computing from Brunel University. His email address is n.mustafee@exeter.ac.uk.

Varun Ramamohan is an Assistant Professor in the Department of Mechanical Engineering at the Indian Institute of Technology Delhi. He holds a Ph.D. in Industrial Engineering from Purdue University. His email address is varunr@mech.iitd.ac.in.