# From black box to white box: automated generation of mathematical programming representation for discrete event simulation models

Mengyi Zhang[a], Arianna Alfieri[b], Andrea Matta[a]

[a] Department of Mechanical Engineering, Politecnico di Milano, via La Masa 1, 20156 Milano, Italy
[b] Department of Management and Industrial Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Turin, Italy

**ABSTRACT**
This work proposes a mathematical programming (MP) representation of discrete event simulation of timed Petri nets (TPN). Currently, MP techniques are not widely applied to optimizing discrete event systems, and this work connects the two fruitful research fields. In the MP formalism, the decision variables are composed of the transition firing time and the markings, while the constraints represents the state transition logic and temporal sequences among events. The MP model and a simulation run of the TPN are totally equivalent, which is validated with several cases. Using a TPN model as input, the MP model can be routinely and conveniently generated.

**KEYWORDS**
discrete event simulation, mathematical programming, timed Petri net

## 1. Introduction

Discrete event dynamic systems (DEVS) has wide application in manufacturing and service applications. Even though many theoretical and analytical studies on discrete event dynamic systems have been developed, performance of real systems sometimes can only be evaluating with discrete event simulation (DES) due to system complexity. Hence, simulation–optimization algorithms are widely used when performance evaluation must be coupled with optimization, i.e., when the best system configuration, according to some criteria, has to be found meanwhile guaranteeing a given value of some performance measure.

Most of the simulation–optimization studies consider DES as a *black–box* function. Under the black–box setting, simulation module and optimization module are decoupled, except that simulation module needs to provide the performance measures of certain solutions when the optimization module queries. With quite limited information from simulation, the optimization module must have many trials, so the black–box procedure can be computationally inefficient.

A promising direction to improve the inefficiency of black-box procedure is to merge the system dynamics into optimization, i.e., considering DES as /textitwhite-box and

CONTACT Mengyi Zhang. Email: mengyi.zhang@polimi.it

modeling system dynamics as part of mathematical programming (MP) models. Several works can be found in literature. An MP model was proposed to minimize the makespan of single-server manufacturing systems in Di Marino, Su, and Basile (2020). The buffer allocation problem in multi-stage production flow lines with blocking is formulated as MP and solved in Matta (2008), Weiss and Stolletz (2015) and Alfieri, Matta, and Pastore (2020). The optimization of systems with complex blocking mechanisms, for instance extended kanban, is addressed in Pedrielli, Alfieri, and Matta (2015) with MP techniques. Decomposition or linear approximation approaches are applied to solving the MP models, and it can be rather efficient than black-box algorithms and the optimality is also enhanced (Weiss and Stolletz (2015), Alfieri et al. (2020)).

However, white-box simulation optimization approaches are not as widely-applied as block-box ones. The reason is that developing a white-box algorithm requires one to master discrete event dynamic systems, its simulation and mathematical programming, but those disciplines are not tightly connected. The above-mentioned works are all tailored to specific cases, and there is a lack of generalization to extend the works to other problems. To fill this gap, this work proposes an approach to translating the simulation of general timed Petri net (TPN) into MP models.

In literature, a few works address the similar problem, i.e., translating the behavior of general discrete event dynamic systems into MP. Bemporad and Morari (1999) proposed a mixed integer programming modeling framework for hybrid systems, whose states are mixed integer, in discrete time. Differently, this work deals with continuous–time discrete–state systems. Chan and Schruben (2008) proposed a modeling framework to translate a DES model into an MP model in a general sense based on the Event Relationship Graph (ERG) of the system dynamics. However, the ERG is not as commonly-used as TPN, and the approach is not used often.

The benefits of using MP model to present discrete event systems are many. As already mentioned above, when coupled with optimization, the MP-based algorithms can be faster and gain better solution than black–box optimization algorithms. Furthermore, black–box approaches have limited capability is solving constrained optimization problems, but MP-based approaches can easily deal with them Zhang and Matta (2020). Thirdly, the vast theoretical and methodological results developed in the MP field can be introduced into the study of DEVS through simulation. For instance, using sensitivity analysis of MP models, DES can be a gradient estimation tool, as suggested in Chan and Schruben (2008). In fact, we do not proposed to totally replace simulation with MP, but to enrich the toolbox for analyzing and optimizing DEVS. Finally, the application of MP models of DEVS is not limited to optimization. Basile, Chiacchio, and De Tommasi (2012) proposed the sufficient and necessary condition of K-diagnosability of TPN based on MP.

The major concern about the application of MP is the computational complexity, and the solving procedure can be time-consuming or even unbearable. However, many approaches from optimization community are available to improve the efficiency. Linear programming approximation (Alfieri and Matta (2012)), Benders decomposition (Weiss and Stolletz (2015)), row-column generation (Alfieri et al. (2020)), have been studied to solve optimization problems in reality based-on MP model of DEVS.

The contributions of this work is to propose the first framework translating general TPN into MP models. The translation procedure can be conveniently implemented in general–purpose programming languages. This work lays a foundation of developing MP-based approaches for analyzing and optimizing general TPN, based on the vast literature in MP field.

The rest of the paper is organized as follows. Section 2 briefly introduce the TPN and its simulation algorithm. Section 3 describes the generation of the MP of a TPN model. Section XXX shows several examples of application and validation of the equivalence of simulation and the MP. Discussion and conclusion are reported in Section XXX and XXX, respectively.

## 2. Simulating TPN

### *2.1. Introduction to timed Petri nets*

A TPN is a directed bipartite graph. The set of nodes in that graph is partitioned into a set of *places* $P$ and a set of *transitions* $T$. Places and Transitions are connected with weighted and directed edges. As in the right side of Figure 1, places (i.e., $p_1$, $p_2$, $p_3$), transitions (i.e., $t_1$, $t_2$) and tokens are represented by white circles, rectangles and small black circles, respectively. Places hold *tokens*, and tokens are transferred through transition *firings*, i.e., one transition firing absorbs tokens from some places and releases to some other places. Each transition $t \in T$ can absorb tokens from all of its precedents, and the number of tokens absorbed from place $p \in P$ is equal to the weight of the edge connecting $p$ and $t$. The number of tokens in each place must be non-negative at any time, hence, a firing of transition $t$ is enabled if and only if the number of tokens in all its precedents $p$ is not smaller than the weight on edge $(p, t)$. Each transition $t \in T$ can release tokens to all of its successors, and the number of tokens released to place $p \in P$ is equal to the weight of the edge connecting $t$ and $p$. It is not necessary that the set of precedents and the set of successors are mutually exclusive. In a TPN, the duration between the moment of absorption and the moment of release is not always negligible, and the duration is called *firing time*. This work considers firing time non-negative and that can follow arbitrary stochastic distribution. The distribution of tokens in the places is called *markings*. The initial state of the system, i.e., the state at time zero, is represented by *the initial marking*. Formally, a TPN can be defined as a 6-tuple $N = (P, T, F, W, M_0, \tau)$, where

$P$ denotes the set of places;
$T$ denotes the set of transitions;
$F$ $\subseteq (P \times T) \cup (T \times P)$ denotes the set of edges;
$W:$ $F \to \mathbb{Z}^+$ denotes the weights on edges;
$M_0:$ $P \to \mathbb{N}$ denotes the initial marking;
$\tau:$ $T \to$ *random distribution* denotes the firing time of transitions.

An illustrative example using G/G/2 queue is showed in Figure 1. In a G/G/2 queue, customers arrive at the queue following an arbitrary arrival process. One of the two identical servers (or the idle one if only one is idle) processes server processes customers in the queue one by one, and the processing time is arbitrarily distributed. After processing, the customer will be released from the system immediately, and the server becomes idle. On the right side of Figure 1, the graph of the TPN of G/G/2 is presented. A token in place $p_1$ shows that it is possible to schedule the arrival of next customer, and the firing time of transition $t_1$ is equal to the inter-arrival time. At the finish of firing $t_1$, a customer arrives, and it is possible to schedule the next arrival, thus, it releases one token to $p_1$. Furthermore, an arrival will increase the length of the queue by 1, which is represented by the number of tokens in place $p_2$. Two tokens in place $p_3$ shows that two servers are idle. The firing of transition $t_2$ shows the processing of a customer, and it is enabled if and only if at least one server is idle and there is

a customer in the queue. Once the process of the customer finishes, i.e., the finish of firing $t_2$, a server is released, and a token is sent to place $p_3$.

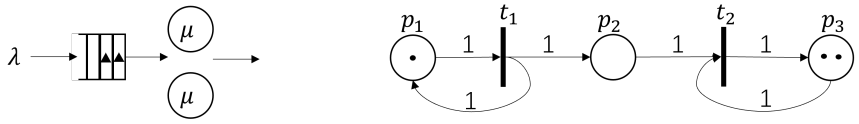


**Figure 1.** G/G/2 queue and its TPN model.

### *2.2. Simulating TPNs*

The event-scheduling approach is the logic behind all the major DES software and used by practitioners when developing simulation codes with general purpose languages (Law (2014)). For sake of completeness, the logic for simulating TPN is briefly described. Within the event scheduling worldview of discrete event simulation (Zeigler, Muzy, and Kofman (2018)), an event is first scheduled and occurs afterwards. An *event list* stores all the scheduled events that will occur in the future. The event with the earliest occurring time is polled to occur from the event list, and the clock is advanced to its occurring time. The system state is changed upon the event occurrence, and the new state enables the schedule of new events. A new cycle starts from polling the earliest event. In TPN, events, whose occurrence changes the system state instantaneously, are either start or finish of transition firings in a TPN, since they are the only moments when the marking is changed. Thus, an event in a simulation realization of TPN can be represented by a triple $(t, type, time)$, where $t$ is a transition, *type* is either *start* or *finish*, and *time* is the occurring time. Given a TPN $N = (P, T, F, W, M_0, \tau)$, its simulation can be implemented as in Algorithm 1.

In Algorithm 1, event list, clock time and system state (i.e., the markings) are initialized as empty, zero and initial markings, respectively. Each iteration is composed of two steps, i.e., to start all the enabled transitions and to finish the firing in the event list with the earliest finishing time. In the first step, if firing of a certain transition $t_1$ is enabled by the current markings, the firing can be started and tokens are immediately removed from precedent places of $t$. In the mean time, the finish of the firing can be scheduled with a delay equal to the firing time, which is a sample $\tau_0$ of random distribution $\tau(t_1)$, and the finish of firing is added to the event list. It can be noticed that the event list contains only finish-of-firing events, since all the start-of-firing events are executed immediately without going into the list. In the second step, the firing with the earliest finish time will be taken from the event list, and the tokens are released to the successive places. The iterative procedure stops when a given condition is reached, which is usually a limit of iteration numbers or clock time.

## 3. MPR models

This section introduces the MPR translating the dynamics of the TPN, equivalent to Algorithm 1. Specifically, the time when transition firings start and finish and markings can all be seen in the MPR as decision variables. The time when transition $t$ starts to fire for the $i$-th time is denoted by $e_i^{t,s}$, and $e_i^{t,f}$ denotes the time when the fire finishes. Variables $\mathcal{E}_k$ denotes the simulation clock at the beginning of iteration $k$ in Algorithm

**Algorithm 1** Simulating a TPN.

**Input:**

Initial markings: $M_0$.

1: **Initialization:**
2: $EventList = \{\}$; simulation clock: $clock \leftarrow 0$; current markings: $M \leftarrow M_0$;
3: iteration counter: $k \leftarrow 0$.
4: **while** stopping condition is not true **do**
5: **for** each transition $t_1 \in T$ **do**
6: **while** current marking enables the firing of $t_1$, i.e., $M(p_1) \geq W(p_1, t_1),\ \forall (p_1, t_1) \in F$ **do**
7: Sampling the firing time $\tau_0$ of random variable $\tau(t_1)$
8: **for** each precedent place $p_1$ of $t_1$ **do**
9: Remove $W(p_1, t_1)$ tokens from $p_1$: $M(p_1) \leftarrow M(p_1) - W(p_1, t_1)$
10: **end for**
11: Add event $(t_1, finish, clock + \tau_0)$ to event list.
12: **end while**
13: **end for**
14: From *EventList*, take the event $(t_2, finish, time)$ with the earliest time.
15: Advance the clock: $clock \leftarrow time$.
16: **for** each successive place $p_2$ of $t_2$ **do**
17: Add $W(t_2, p_2)$ tokens to $p_2$: $M(p_2) \leftarrow M(p_2) + W(t_2, p_2)$.
18: **end for**
19: Update iteration counter: $k = k + 1$
20: **end while**

1. Variables $e_i^{t,s}$, $e_i^{t,f}$ and $\mathcal{E}_k$ are all real-valued and non-negative. Variable $m_k^p$ is used to denote the number of tokens in place $p$ at the beginning of iteration $k$ in Algorithm 1. Some binary variables are also used in the MPR, and they will be introduced in the following, during the explanation of the model.

### *3.1. Event execution constraints*

The first group of mathematical relationships, denoted by group-A, are the constraints for firing finishes, as in lines 14 and 15 in Algorithm 1. Binary variables $w_{i,k}^t$ are introduced to represent that the $i$-th start firing of transition $t$ finishes in iteration $k$. If $w_{i,k}^{\xi}$ is equal to one, and variable $e_i^{t,f}$ is binded to $\mathcal{E}_k$, as shown in constraints (A1) and (A2). Constraints (A3) state that each firing-finish event executes at most once. Constraints (A4) state that in each iteration, one firing-finish event is executed. Constraints (A5) imply that the clock cannot be reversed from iteration $k-1$ to iteration $k$. Constraint (A6) implies that the simulation clock is initialized to zero. Constraints (A7) implies that the delay between the start and finish of firing transition $t$ is equal to a sample from the random variate $T^t$.

$$e_i^{t,f} - \mathcal{E}_k \geq M(w_{i,k}^t - 1) \quad \forall\, t \in \mathbb{T}, i \in \mathbb{I}^t, k \in \mathbb{K} \quad (A1)$$

$$\mathcal{E}_k - e_i^{t,f} \geq M(w_{i,k}^t - 1) \quad \forall\, t \in \mathbb{T}, i \in \mathbb{I}^t, k \in \mathbb{K} \quad (A2)$$

$$\sum_{k \in \mathbb{K}} w_{i,k}^t \leq 1 \qquad \forall\, t \in \mathbb{T}, i \in \mathbb{I}^t \qquad (A3)$$

$$\sum_{t \in \mathbb{T}} \sum_{i \in \mathbb{I}^t} w_{i,k}^t = 1 \qquad \forall\, k \in \mathbb{K} \qquad (A4)$$

$$\mathcal{E}_k - \mathcal{E}_{k-1} \geq 0 \qquad \forall\, k \in \mathbb{K} \qquad (A5)$$

$$\mathcal{E}_0 = 0 \qquad (A6)$$

$$e_i^{t,f} - e_i^{t,s} = \tau_i^t \qquad \forall t \in \mathbb{T}, i \in \mathbb{I}^t \qquad (A7)$$

### *3.2. To start firing*

The second group of constraints, denoted by group B, depict the start of transition firing, as in lines 5 to 13 in Algorithm 1. Binary variables $x_{i,k}^t$ represent that a fire of transition $t$ starts in iteration $k$ if it is equal to one. Constraints (B1) and (B2) shows that the start firing time is equal to the simulation clock.

$$e_i^{t,s} - \mathcal{E}_k \geq M(x_{i,k}^t - 1) \quad \forall\, t \in \mathbb{T}, k \in \mathbb{K}, i \in \mathbb{I}^t \quad (B1)$$

$$\mathcal{E}_k - e_i^{t,s} \geq M(x_{i,k}^t - 1) \quad \forall\, t \in \mathbb{T}, k \in \mathbb{K}, i \in \mathbb{I}^t \quad (B2)$$

The condition to start firing transition $t$ is that the tokens in all the precedent places $p$ are above the required level $A^{p,t}$. Binary variables $z_k^t$ equal to one represents the condition for transition $t$ is true, as constraints (B3). Moreover, a set of binary variables $v_k^{t,p}$ are used to verify if tokens in precedent place $p$ is smaller than $W^{p,t}$, as constraints (B4). Constraints (B5) assures that $z_k^t$ is equal to zero only if at least one precedent place does not contain enough number of tokens. Constraints (B6) show that if $z_k^t$ is equal to one, transition $t$ must start to fire in iteration $k$. Constraints (B7) state that at most one firing of any transition starts for each transition.

$$m_k^p - W^{p,t} \geq M(z_k^t - 1) \quad \forall\, t \in \mathbb{T}, k \in \mathbb{K}, p \in \mathbb{P} \quad (B3)$$

$$(W^{p,t} - 1) - m_k^p \geq M(v_k^{t,p} - 1) \quad \forall\, t \in \mathbb{T}, k \in \mathbb{K}, p \in \mathbb{P} \quad (B4)$$

$$1 - z_k^t \leq \sum_{p \in \mathbb{P}} v_k^{p,t} \qquad \forall\, t \in \mathbb{T}, k \in \mathbb{K} \qquad (B5)$$

$$\sum_{i \in \mathbb{I}^t} x_{i,k}^t = z_k^t \qquad \forall\, t \in \mathbb{T}, k \in \mathbb{K} \qquad (B6)$$

$$\sum_{k \in \mathbb{K}} x_{i,k}^t \leq 1 \qquad \forall\, t \in \mathbb{T}, i \in \mathbb{I}^t \qquad (B7)$$

However, certain markings can enable firings of the same transition to start multiple times, which is forbidden by constraints (B7). Thus, modifications are made to the TPN model before the proposed MPR model is applied. The modification is to expand the non-zero-firing-time transition $t$ into two transitions, denoted by $\tilde{t}$ and $\bar{t}$,

respectively. Transition $\tilde{t}$ has zero firing time and maintains all the precedent places and their weights of transition $t$, while transition $\bar{t}$ maintains the firing time, successive place and their weights of transition $t$. Transitions $\tilde{t}$ and $\bar{t}$ are both connected to an intermediate place $p_t$, and with one arc pointed from $\tilde{t}$ to place $p_t$ and one arc pointed from $p_t$ to $\bar{t}$. An example is shown in Figure 2. This expansion can freeze the simulation clock before all the possible firings of transition $t$ start.


**Figure 2.** TPN expansion.

### *3.3. Event sequencing constraints*

The index $i$ of event $e_i^{t,s}$ represents the sequence of firing starting, i.e., if a firing starts in an earlier iteration, then its index will be smaller. Moreover, a firing must finish after starting. Group-C constraints force such sequences, which are relevant to all events. Constraints (C1) show that if the $i$-th firing of transition $t$ does not start before simulation termination, then the $(i+1)$-th firing will not start. Constraints (C2) state that if the $i$-th firing of transition $t$ does not start before simulation termination, then it will not finish. Constraints (C3) state that a firing cannot finish before starting, unless it remains in the future event list at the end of the simulation run. Constraints (C4) depict that the $(i+1)$-th firing of transition $t$ must be scheduled after the $i$-th execution, unless the $(i+1)$-th execution is not scheduled before simulation termination.

$$\sum_{k\in\mathbb{K}} x_{i+1,k}^t - \sum_{k\in\mathbb{K}} x_{i,k}^t \leq 0 \quad \forall\, t \in \mathbb{T}, i \in \mathbb{I}^t \quad (C1)$$

$$\sum_{k\in\mathbb{K}} w_{i,k}^t - \sum_{k\in\mathbb{K}} x_{i,k}^t \leq 0 \quad \forall\, t \in \mathbb{T}, i \in \mathbb{I}^t \quad (C2)$$

$$\sum_{k\in\mathbb{K}} k w_{i,k}^t - \sum_{k\in\mathbb{K}} k x_{i,k}^t \geq M(\sum_{k\in\mathbb{K}} w_{i,k}^t - 1) \quad \forall\, t \in \mathbb{T}, i \in \mathbb{I}^t \quad (C3)$$

$$\sum_{k\in\mathbb{K}} k x_{i+1,k}^t - \sum_{k\in\mathbb{K}} k x_{i,k}^t \geq 1 + M(\sum_{k\in\mathbb{K}} x_{i+1,k}^t - 1) \quad \forall\, t \in \mathbb{T}, i \in \mathbb{I}^t \quad (C4)$$

### *3.4. State transitions*

The tokens in place $p$ are changed from $m_k^p$ to $m_{k+1}^p$ in iteration $k$, as in constraints (E1) and in lines 16 to 18 in Algorithm 1. The transitions $t$ that start in fire in iteration $k$ absorbs $A^{p,t}$ tokens from the precedent places $p$, and the firing that finish in iteration $k$ releases tokens to the successive places.

$$m_{k+1}^p = m_k^p - \sum_{t\in\mathbb{T}} W^{p,t} \sum_{i\in\mathbb{I}^t} x_{i,k}^t + \sum_{t\in\mathbb{T}} W^{t,p} \sum_{i\in\mathbb{I}^t} w_{i,k}^t \quad \forall p \in \mathbb{P}, k \in \mathbb{K} \quad (E1)$$

### *3.5. Objective function*

With the constraints defined in the previous sections, there is a unique feasible solution in terms of transition firing times and the history of simulation clock, i.e., $e_i^{t,s}$, $e_i^{t,f}$ and $\mathcal{E}_k$. Thus, the objective function can be any function of those variables, for instance, average of system time or waiting time in queueing systems. Multiple feasible solutions may appear in terms of binary variables, since the sequence of events with identical execution time is not uniquely defined.

The flexibility of the objective function definition is a main difference between the formulation proposed by Chan and Schruben (2008) and this work, since the objective function of MPR in Chan and Schruben (2008) can be only the sum of all execution times. The uniqueness of optimal solution and flexibility of objective function formulation is particularly beneficial if one wants to make use of the resulting MPR to solve an optimization problem concerning the design or operation of the discrete event system (for instance the capacity of the queue or the control policy of a manufacturing system), since changing the objective function or adding new constraints to calculate the system performance will not influence the equivalence of MPR and a simulation run.

## 4. Applications

In this section, several examples are presented, including a G/G/m queue, a merge queueing system composed by three single server stations, and a single server queue with failure as an example of event cancellation. The equivalence of MPR solution and history of a simulation run has been validated with $K$ equal to 20 for 100 independent replicates.

### *4.1. G/G/m queue*

The first example is a G/G/m queue, and the TPN is shown in Figure 3. In a G/G/m queue, customers arrive at the queue following an general arrival process. One of the identical idle servers processes customers in the queue one by one, and the processing time is generally distributed. After processing, the customer will be released from the system immediately, and the server becomes idle again. In Figure 3, the TPN of the TPN of G/G/2 is presented. A token in place $p_{arr}$ shows that it is possible to schedule the arrival of next customer, and the firing time of transition $t_{arr}$ is equal to the inter-arrival time. At the end of firing $t_{arr}$, a customer arrives, and it is possible to schedule the next arrival, thus, it releases one token to $p_{arr}$. Furthermore, an arrival will increase the length of the queue by 1, which is represented by the number of tokens in place $p_{queue}$. Two tokens in place $p_{idle}$ shows that two servers are idle. The firing of transition $t_{process}$ shows the processing of a customer, and it is enabled if and only if at least one server is idle and there is a customer in the queue. Once the process of the customer finishes, i.e., the end of firing $t_{process}$, a server is released, and a token is sent to place $p_{idle}$.

As mentioned in Section 3.2, the transitions with non-zero firing times are expanded before implementing the MPR generation. The expanded TPN of the G/G/2 queue is shown in Figure 4. Transition $t_{process}$ is expanded to $\tilde{t}_{process}$, $\bar{t}_{process}$ and $p_{process}$. Even though transition $t_{arr}$ also has non-zero firing time, it is self-limiting, i.e., there is never more than one firing in process. Therefore, it is not necessary to expand transition $t_{arr}$.

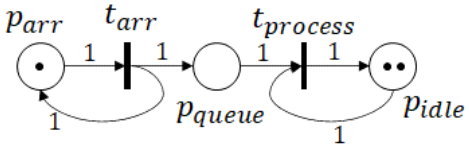


**Figure 3.** TPN of G/G/2.

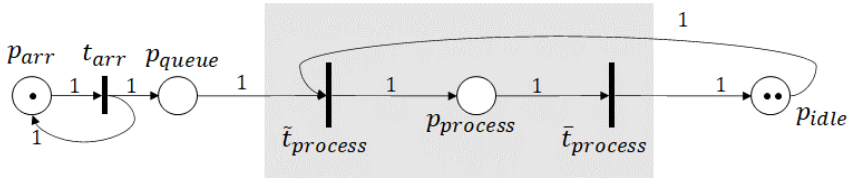


**Figure 4.** TPN expansion of G/G/2.

The MPR of G/G/m queue is presented in the follows. The set of transitions $\mathbb{T}$ is specified by $\{t_arr, t_queue, \tilde{t}_{process}, \bar{t}_{process}\}$ for G/G/m queue. The constraints

Table 1 shows a simulation run of a G/G/m queue including 10 executions. In iteration zero, the system is in initial state, and this state satisfies the condition to schedule the counting event of arrival, as in lines 2 to 8 in Algorithm 3. In the solution of MPR, the scheduling of $a\tilde{r}r$ is represented by $z_0^{a\tilde{r}r}$ and $x_{1,0}^{a\tilde{r}r}$ equal to 1, and its occurring time is equal to the simulation clock, which is equal to 0. The execution $(a\tilde{r}r, 1, e_1^{a\tilde{r}r,0}, e_1^{a\tilde{r}r,1})$, denoted by $e_1^{a\tilde{r}r}$ in the table, is then performed since it is the earliest execution (line 18 in Algorithm 3), which is also indicated by $w_{1,1}^{a\tilde{r}r}$ equal to one, and simulation clock in the next iteration, i.e., iteration 1, is equal to the occurring time $e_{1,1}^{a\tilde{r}r}$ in the solution of MPR. After that, the positive-delay event $arr$ is then scheduled (lines 26 to 30 in Algorithm 3), which can also be seen from the MPR solution, since $x_{1,1}^{arr}$ is equal to one, the scheduling time of $e_{1,0}^{arr}$ is equal to the current simulation clock time and the occurrence time is 2.3 time unit later than the scheduling time, where 2.3 is sampled from the distribution of inter-arrival time $T^{arr}$. At the end of the iteration, the system state is updated according to the state transition function of event $a\tilde{r}r$ (line 25 in Algorithm 3), which in the MPR solution is indicated by an increment equal to one of state variable $u^{arr}$. This procedure is repeated in the next iterations, and the equivalence of DES procedure and the solution of MPR can be seen from Table 3.

**Table 1.** Simulation run and MPR solution of G/G/m queue.

| $k$ | *clock* | Markings | Start firings | Unfinished firings | Finish Firing |
|---|---|---|---|---|---|
| 0 | 0 | $(1,0,0,2)$ | $t_{arr}$ | $t_{arr}:2.3$ | $t_{arr}:2.3$ |
| | $\mathcal{E}_0=0$ | $\mathbf{m}_0=(1,0,0,2)$ | $z_0^{arr}=1, x_{1,0}^{arr}=1, e_1^{arr,s}=0, e_1^{arr,f}=2.3$ | | $w_{1,0}^{arr}=1$ |
| 1 | 2.3 | $(1,1,0,2)$ | $t^{arr}, \tilde{t}_{proc}$ | $\tilde{t}_{proc}:2.3, t^{arr}:11.1$ | $\tilde{t}_{proc}:2.3$ |
| | $\mathcal{E}_1=2.3$ | $\mathbf{m}_1=(1,1,0,2)$ | $z_1^{arr}=1, x_{2,1}^{arr}=1, e_2^{arr,s}=2.3, e_2^{arr,f}=11.1$ | | $w_{1,1}^{\tilde{proc}}=1$ |
| | | | $z_1^{\tilde{proc}}=1, x_{1,1}^{\tilde{proc}}=1, e_1^{\tilde{proc},s}=2.3, e_1^{\tilde{proc},f}=2.3$ | | |
| 2 | 2.3 | $(0,0,1,1)$ | $\bar{t}_{proc}$ | $\bar{t}_{proc}:6, t^{arr}:11.1$ | $\bar{t}_{proc}:6$ |
| | $\mathcal{E}_2=2.3$ | $\mathbf{m}_2=(0,0,1,1)$ | $z_2^{\bar{proc}}=1, x_{1,2}^{\bar{proc}}=1, e_1^{\bar{proc},s}=2.3, e_1^{\bar{proc},f}=6$ | | $w_{1,2}^{\bar{proc}}=1$ |
| 3 | 6 | $(0,0,0,2)$ | | $t^{arr}:11.1$ | $t^{arr}:11.1$ |
| | $\mathcal{E}_3=6$ | $\mathbf{m}_3=(0,0,0,2)$ | | | $w_{2,3}^{arr}=1$ |
| 4 | 11.1 | $(1,1,0,2)$ | $t^{arr}, \tilde{t}^{proc}$ | $\tilde{t}^{proc}:11.1, t^{arr}:12.1$ | $\tilde{t}_{proc}:11.1$ |
| | $\mathcal{E}_4=11.1$ | $\mathbf{m}_4=(1,1,0,2)$ | $z_4^{arr}=1, x_{3,4}^{arr}=1, e_3^{arr,s}=11.1, e_3^{arr,f}=12.1$ | | $w_{2,4}^{\tilde{proc}}=1,$ |
| | | | $z_4^{\tilde{proc}}=1, x_{2,4}^{\tilde{proc}}=1, e_2^{\tilde{proc},s}=11.1, e_2^{\tilde{proc},f}=11.1$ | | |
| 5 | 11.1 | $(0,0,1,1)$ | $\bar{t}_{proc}$ | $t^{arr}:12.1, \bar{t}_{proc}:16.9$ | $t^{arr}:12.1$ |
| | $\mathcal{E}_5=11.1$ | $\mathbf{m}_5=(0,0,1,1)$ | $z_5^{\bar{proc}}=1, x_{2,5}^{\bar{proc}}=1, e_2^{\bar{proc},s}=11.1, e_2^{\bar{proc},f}=16.9$ | | $w_{3,5}^{arr}=1$ |
| 6 | 12.1 | $(1,1,0,1)$ | $t^{arr}, \tilde{t}^{proc}$ | $\tilde{t}^{proc}:12.1, t^{arr}:15.2, \bar{t}_{proc}:16.9$ | $\tilde{t}^{proc}:12.1$ |
| | $\mathcal{E}_6=12.1$ | $\mathbf{m}_6=(1,1,0,1)$ | $z_6^{arr}=1, x_{4,6}^{arr}=1, e_4^{arr,s}=12.1, e_4^{arr,f}=15.2$ | | $w_{3,6}^{\tilde{proc}}=1$ |
| | | | $z_6^{\tilde{proc}}=1, x_{3,6}^{\tilde{proc}}=1, e_3^{\tilde{proc},s}=12.1, e_3^{\tilde{proc},f}=12.1$ | | |
| 7 | 12.1 | $(0,0,1,0)$ | $\bar{t}^{proc}$ | $t^{arr}:15.2, \bar{t}^{proc}:16.9, \bar{t}^{proc}:20.1$ | $t^{arr}:15.2$ |
| | $\mathcal{E}_7=12.1$ | $\mathbf{m}_7=(0,0,1,0)$ | $z_7^{\bar{proc}}=1, x_{3,7}^{\bar{proc}}=1, e_3^{\bar{proc},s}=12.1, e_3^{\bar{proc},f}=20.1$ | | $w_{4,7}^{arr}=1$ |
| 8 | 15.2 | $(1,1,0,0)$ | $t^{arr}$ | $\bar{t}^{proc}:16.9, t^{arr}:17.8, \bar{t}^{proc}:20.1$ | $\bar{t}^{proc}:16.9$ |
| | $\mathcal{E}_8=15.2$ | $\mathbf{m}_8=(1,1,0,0)$ | $z_8^{arr}=1, x_{5,8}^{arr}=1, e_5^{arr,s}=15.2, e_5^{arr,f}=17.8$ | | $w_{2,8}^{\bar{t}^{proc}}=1$ |
| 9 | 16.9 | $(0,1,0,1)$ | $\tilde{t}^{proc}$ | $\tilde{t}^{proc}:16.9, t^{arr}:17.8, \bar{t}^{proc}:20.1$ | $\tilde{t}^{proc}:16.9$ |
| | $\mathcal{E}_9=16.9$ | $\mathbf{m}_9=(0,1,0,1)$ | $z_9^{\tilde{proc}}=1, x_{4,9}^{\tilde{proc}}=1, e_4^{\tilde{proc},s}=16.9, e_4^{\tilde{proc},f}=16.9$ | | $w_{4,9}^{\tilde{proc}}=1$ |
| 10 | 16.9 | $(0,0,1,0)$ | $\bar{t}^{proc}$ | $t^{arr}:17.8, \bar{t}^{proc}:20.1, \bar{t}^{proc}:25.5$ | $t^{arr}:17.8$ |
| | $\mathcal{E}_{10}=16.9$ | $\mathbf{m}_{10}=(0,0,1,0)$ | $z_{10}^{\bar{proc}}=1, x_{4,10}^{\bar{proc}}=1, e_4^{\bar{proc},s}=16.9, e_4^{\bar{proc},f}=25.5$ | | $w_{5,10}^{arr}=1$ |

## 5. Introduction

Discrete Event Simulation (DES) is one of the most used tools for performance evaluation of complex systems in manufacturing and service applications. Hence, simulation–optimization algorithms are widely used when performance evaluation must be coupled with optimization, i.e., when the best system configuration, according to some criteria, has to be found meanwhile guaranteeing a given value of some performance measure. Most of the state–of–the–art simulation–optimization algorithms consider DES as a *black–box* function, and the structure of DES models has been seldom studied. On the contrary, a minority of the simulation–optimization literature explores the structure of the DES models, and such approaches are referred to as *white–box* simulation–optimization. Under the black–box setting, simulation–optimization algorithms work in an iterative way, alternating simulation and optimization procedures, thus, possibly leading to computational inefficiency if the number of iterations and/or the computation time per iteration increases too much. The benefit of white–box simulation–optimization is the saving of simulation budget or computation time since the optimization procedure is guided by the information contained in the structure of the DES model (Zhang, Matta, and Alfieri (2020)). However, the barrier to the use of white–box simulation–optimization is modeling DES as white box, so that it eventually favors optimization.

The benefit of developing an MPR might be not obvious (especially when there is already a DES model) due to the extremely high complexity of solving it. However, this work does not suggest solving the MPR directly with state-of-the-art optimizers. Instead, the optimal solution of the MPR can be obtained from a simulation run. The added value of the MPR is that the structure of the DES can be explored. With the vast theoretical and methodological results developed in the mathematical programming (MP) field, for instance sensitivity analysis as suggested by Chan and Schruben (2008), the MPR of a simulation model favors the optimal design and control of the discrete event systems.

This work proposes a procedure to establish a white–box simulation model, which is an equivalent Mathematical Programming Representation (MPR) model, based on the well-known event–scheduling logic (Law (2014)). The procedure is applicable to certain types of DES models, and the assumptions that the DES model should satisfy are also presented in this work.

Chan and Schruben (2008) proposed a modeling framework to translate a DES model into an MPR model in a general sense. Their modeling framework is based on the Event Relationship Graph (ERG) of the system dynamics. To derive the MPR, an ERG of the DES must be constructed and expanded to an elementary ERG (EERG) model, and a procedure can be applied to translate the EERG model into an MPR. However, this procedure has some limitations. First, deriving an ERG is not an easy task, and the user must pay quite much attention to detect all the event relationships and complete the triggering conditions between each pair of related events. The difficulty of developing ERG limits the wide spread of this procedure. Second, the modeling procedure is case–by–case depending on the event relationships, which means that the user must analyze the event relationships one by one and identify which type of modeling, including the variables and constraints, he/she should apply for each event relationship. Consequently, an automatic procedure was not proposed. This is quite difficult, since EERG is an expansion of ERG; the resulting graph could be huge and writing down the complete MPR model could be even impossible.

This work proposes a procedure that does not need the ERG and can be used to

automatically generate the MPR in a general–purpose programming language. Despite being different, the MPRs proposed in this work and Chan and Schruben (2008) lead to equivalent results, which, in turn, are both equivalent to a simulation realization. The procedure can also be applied to devise MPR of DES with event cancellation, which was not considered in the work of Chan and Schruben (2008).

Many works in the literature show the potentiality of this research direction. For instance, the gradient can be conveniently estimated from the simulation model, if the MPR is approximated into Linear Programming (LP) and the dual can be conveniently obtained (Chan and Schruben (2008)). Moreover, if some of the parameters in the MPR are changed to decision variables, the MPR becomes an integrated simulation–optimization model. Solving the integrated model provides the optimal solution of the optimization problem (Matta (2008)). MP–based algorithms, such as linear programming approximation (Alfieri and Matta (2012)), Benders decomposition (Weiss and Stolletz (2015)), row-column generation (Alfieri et al. (2020)), have been applied to improve the efficiency of integrated MP model solution.

The application of MPR–based simulation–optimization approaches is usually found in operations management of manufacturing and service systems. The flexibility of DES for complex system evaluation and of MPR for modeling optimization problems, allowed the integrated simulation-optimization approach to be effectively applied to buffer allocation problems (Zhang, Pastore, Matta, and Alfieri (2021)), even with complex blocking mechanisms (Pedrielli et al. (2015)) and to problems involving real value decision variables (Tan (2015); Zhang and Matta (2020)). Before the above-mentioned works were proposed, there were many state–of–the–art heuristic approaches addressing those problems, but without any guarantee of global or local optimality. Thus, the development of MPR–based simulation–optimization has made its contribution in the research area of manufacturing and service system optimization.

The contributions of this work are several and refer to different aspects. First, it proposes an MPR of DES from event-scheduling execution logic, which is the foundation of many DES implementations. Thus, the procedure does not require much extra effort once one has an event-scheduling execution logic implemented, while the state-of-the-art approach requires the DES modeled with ERG. The proposed approach can be easily automated (the source code has been uploaded in a repository, as explained in the following). Second, it proposes the MPR of event cancellation, which is usually used to model queue with abandonment or servers with failures but has never been studied in the literature. This work will be a foundation of developing math-programming-based simulation-optimization algorithm, based on the vast literature in mathematical programming filed, for instance, the gradient estimation. The proposed MPR can also be easily transformed into an MPR for optimizing the design parameters of the DES, which are common optimization problems in operations management field.

The rest of the paper is organized as follows. Section 6 describes the generation of the MPR of a DES model, including a brief introduction of event-scheduling algorithm, the assumptions for applying the proposed procedure, a modified event-scheduling algorithm, and the MPR itself automatically generated based on the model. Section 7 shows several examples of DES and the generated MPR, whose equivalence has been validated. Result discussion and conclusion are reported in Section 8 and 9, respectively.

## 6. MPR generation procedure

### *6.1. Event-scheduling execution logic of DES*

The event-scheduling approach is the logic behind all the major DES software and used by practitioners when developing simulation codes with general purpose languages (Law (2014)). For sake of completeness, the logic is briefly described. The fundamental elements are the *system state* and the *events*. The system state $\mathbf{s}$ is a set of state variables $s$ needed to describe the system at a particular time. An event $\xi$ is composed by a collection of four objects, $(CS^{\xi}(\mathbf{s}),\ CC^{\xi}(\mathbf{s}),\ F^{\xi}(\mathbf{s}),\ T^{\xi})$, where:

$$\begin{aligned} CS^{\xi}(\mathbf{s}): &\quad \text{condition to schedule} \\ CC^{\xi}(\mathbf{s}): &\quad \text{condition to cancel} \\ F^{\xi}(\mathbf{s}): &\quad \text{state transition function} \\ T^{\xi}: &\quad \text{delay between schedule and occurrence.} \end{aligned}$$

Condition to schedule and to cancel an event are logic expressions based on the system state $\mathbf{s}$, and the output will be true if the system state allows to schedule or cancel executions. The occurrence of an event changes the system state, which is indicated by the state transition function. The input and output of $F^{\xi}(\mathbf{s})$ are the system state before and after event occurrence, respectively. There is a delay, zero or positive, between scheduling time and occurring time, and this delay is called *occurrence delay* and denoted as $T^{\xi}$ in the rest of the paper. In this work, $T^{\xi}$ is considered as random variate. Generally, each event usually occurs more than once during a simulation run, and the term *execution* is introduced to specify a certain occurrence of an event. An execution is composed by a set of four objects, $(\xi, i, \tau_0, \tau_1)$, where $\xi$ denotes the event type, $i$ denotes the index of execution, $\tau_0$ denotes the scheduling time and $\tau_1$ denotes the occurring time.

Given a set of events $\mathbb{X}$, initial system state $\mathbf{s}_0$, and simulation length defined as the number of iterations and denoted by $K$, a DES model could be implemented with the event-scheduling logic, as in Algorithm 2. The algorithm is initialized with simulation the clock equal to zero, number of executions scheduled in the past for each event type equal to zero, and empty future event list. The *future event list* is a list of executions, which are scheduled but not occur yet. Each iteration is composed by three steps, which are scheduling new executions, canceling executions and performing an execution. First, for each event type $\xi$, if the condition to schedule $\xi$ is true, a new execution of $\xi$ is created and added to the future event list, as in lines 3 to 9. Then, for each execution in the future event list, if the condition to cancel the event is true, the executions of that event type is removed from the future event list, as in lines 12 to 16. Finally, the execution with the earliest occurring time is taken from the future event list, the system state is changed according to the state transition function of the event type, and the simulation clock is set to the occurring time. If there are more than one execution with the same earliest occurring time, the algorithm can randomly pick one among them or according to a priority rule.

In the following, a procedure to translate DES models into MPR models, based on the event-scheduling logic, is introduced. Before presenting the procedure, the assumptions that the DES model has to satisfy in order to have the procedure applicable, are introduced.

### *6.2. Assumptions*

To apply the procedure proposed in this work, the following assumptions must be satisfied.

(1) State variables are integer.
(2) For all the event types $\xi$, the *condition to schedule* and the *condition to cancel* are in the form $a^{\xi,s} \leq s \leq c^{\xi,s}$, where $a^{\xi,s}$ and $c^{\xi,s}$ are integers and represent, respectively, the lower bound and the upper bound of a range, for a certain state variable $s$. When multiple state variables are involved, they are combined using the logical operator "AND".
(3) The occurring delay of each execution can be generated off-line.
(4) When more than one execution in the future event list have the same occurring time, the sequence of performing the executions is immaterial, i.e., different sequences lead to the same new system state and the same new event list when the simulation clock is advanced.
(5) Only events with strictly positive occurrence delay can be canceled.

The first assumption requires the state variables to be integer. Integer variables widely exist in DES models, such as number of jobs in buffers, idle servers, and binary variables to model system behavior and control. A discrete state variable can be translated into an integer variable or a set of binary variables. For real-valued state variables, they can be approximately discretized. Thus, the first assumption is fairly general.

The second assumption is, instead, more strict. However, if a model does not satisfy this condition, one can consider to introduce extra binary variables to satisfy assumption (2). For instance, if the condition to schedule executions of event type $\xi$ is $s \leq a^{\xi,s}\ OR\ s \geq c^{\xi,s}$, a binary variable $s^{'}$ can be introduced in the model, and $s^{'}$ is equal to one if and only if $s \leq a^{\xi,s}\ OR\ s \geq c^{\xi,s}$. Thus, violation of assumption (2) can be overcome, even though at the cost of an increase of the model complexity.

As for the third assumption, the delay represents usually inter-arrival time or service time in queueing systems. These delays will be used as the constant term of some constraints in the proposed MPR, hence, they should be generated offline, i.e., independently from the real-time state of the system. For instance, both stationary and non-stationary arrival process will satisfy this assumption.

The fourth assumption, in practice, says that the execution time is the only attribute of priority for the event executions in the future event list. If one would like to specify a different priority for events having the same execution time, he/she can specify the priority by adding the event with lower priority to the event list after the execution of the event with higher priority, which can be done by introducing extra binary variables.

The fifth assumption is an extension of the fourth one. Assumption (4) implies that only events with positive delay can be canceled. Let us assume that there is a zero-delay event $\xi$ with cancellation. After an execution, $(\xi, i, \tau_0, \tau_0, false)$, is added to the future event list in iteration $k$. It is obvious that the condition to schedule and the condition to cancel an event are not identical; thus, either the new execution must occur immediately if there is no other execution in the future event list having the same occurring time $\tau_0$ and event cancellation is not relevant, or there are such executions and two cases can occur. In one case, execution $(\xi, i, \tau_0, \tau_0, false)$ is immediately executed in iteration $(k+1)$. In the other case, if some other executions with occurring time equal to $\tau_0$ are executed earlier than execution $(\xi, i, \tau_0, \tau_0, false)$, and the condition to cancel $\xi$ is true at a certain time, then the execution $(\xi, i, \tau_0, \tau_0, false)$

is canceled and never occurs. The simulation realization in the latter case is different from the former one, and assumption (4) is violated. Thus, only positive-delay event can be canceled.

### *6.3. Modified event-scheduling execution logic*

It is not trivial to generate an MPR based on Algorithm 2. Thus, a modified event-scheduling execution logic is proposed this modified algorithm is equivalent to Algorithm 2, but enables the MPR generation. In the modified algorithm, each event is categorized as zero-delay or positive-delay, depending on whether the occurrence delay $T^{\xi}$ is equal to zero with probability equal to one or not. The modifications differ for zero-delay and positive-delay events.

In Algorithm 2, there may be more than one execution of the event type $\xi$ in the future event list. The first modification is that the number of executions of any zero-delay event in the future event list at any time is no more than one. Multiple executions are equivalent to sequentially schedule a new execution after the previous one occurs.

The second modification is about the scheduling of positive-delay events. A *counting event* $\tilde{\xi}$ and a counting variable $u^{\xi}$ are artificially created. The counting event $\tilde{\xi}$ is zero-delay, with the same scheduling condition as $\xi$. The simulation logic is then modified as follows. When the system state satisfies the condition to schedule $\xi$, an execution of event $\tilde{\xi}$, other than $\xi$, is added to the future event list. Only after event $\tilde{\xi}$ occurs, an execution of event type $\xi$ is added to the future event list. The counting variable $u^{\xi}$ represents the number of executions of $\xi$ in the future event list, and its value is incremented by one when counting event $\tilde{\xi}$ occurs and decremented by one when event $\xi$ occurs.

The third modification is about event cancellation. A fifth object named *cancel* is introduced to an *execution*, which has a binary value indicating if the execution is canceled. It is initialized as *false* (i.e., zero) when an execution is created. Once the condition to cancel executions of event type $\xi$ is true, the value of *cancel* of those executions is changed to *true*, (i.e., one), the counting variable $u^{\xi}$ is set to zero, but the execution is not removed from the future event list. When a canceled execution is taken from the future event list, the system state will remain unchanged and the algorithm will continue to the next iteration.

Based on the assumptions and the three modifications above, compositions of zero-delay and positive-delay events can be modified accordingly. Zero-delay events are composed by a collection of two objects ($CS^{\xi}(\mathbf{s})$, $F^{\xi}(\mathbf{s})$), which are condition to schedule and state transition. Positive-delay events are composed by five objects ($T^{\xi}$, $\tilde{\xi}$, $u^{\xi}$, $CC^{\xi}(\mathbf{s})$, $F^{\xi}(\mathbf{s})$), which are occurrence delay, counting event, counting variable, condition to cancel and state transition, respectively.

The last modification is that besides integer $K$, which represents the total number of iterations, the number of executions of each event $\xi$, denoted by $N^{\xi}$, should also be provided. It is not necessary that $N^{\xi}$ is exactly equal to the number of executions of event $\xi$ in the simulation run, instead, it could be an upper bound. Generally speaking, $N^{\xi}$ can be equal to K, but the smaller the value, the smaller the number of variables, thus, a simpler model will be consequently developed. Since $N^{\xi}$ is not the exact number of executions of event $\xi$, when the simulation terminates, the future event list sometimes can be not empty, i.e., some executions are scheduled but never occur.

The modified event-scheduling logic is shown in Algorithm 3. The differences be-

tween the two algorithms are the following. In each iteration, only zero-delay events are scheduled by validating the condition to schedule, as in line 3. Positive-delay events are scheduled after the execution of its counting event is performed, as in lines 26 to 30. Canceled executions are not removed from the future event list, but the attribute *cancel* is set to true, and the counting variable is set zero, as in line 13. When performing an execution, only the transition function of uncanceled executions will be applied, as in lines 21 to 23.

To summarize, to implement the modified event-scheduling logic, zero-delay events, including counting events of positive-delay events, must be provided together with the condition to schedule and the transition function, and positive-delay events must be provided together with occurrence delay, counting event, counting variable, condition to cancel and state transition function. An example of G/G/m queue is shown in Table 2. Three events and two state variables are essential. The three events represent job arrival, service start and service finish, and are denoted by $arr$, $ss$ and $sf$, respectively. The two state variables represent the number of jobs in the queue and the number of occupied servers, and are denoted by $q$ and $g$, respectively. State variable $q$ is non-negative integer, and $g$ can be any integer between 0 and $m$, which is the number of servers. Since event cancellation is not relevant to G/G/m queue, the condition to cancel is not showed. Event $arr$ is a positive-delay event, as explained above. A counting event of $arr$, i.e., $a\tilde{r}r$, and a counting variable $u^{arr}$ are defined. The condition to schedule $arr$ is that $u^{arr}$ is equal to zero, since it is a renewal process, i.e., each time a job arrives, one and only one new arrival can be scheduled. When $arr$ is executed, one job arrives to the system, and queue level $q$ is increased by one. Event $ss$ is a zero-delay event. The condition to schedule $ss$ is that there is one job waiting in the queue and one server available. Upon execution of $ss$, queue level is reduced by one, and the number of occupied servers is increased by one. Event $fs$ is a positive-delay event, it is scheduled immediately after $ss$ is executed, and the delay until its execution is equal to the service time, i.e., sampled from $T^{sf}$. Therefore, event $ss$ can be regarded as the counting event of $sf$. Furthermore, state variable $g$ is the counting variable of $sf$. The only information missing in the Table to complete the algorithm is number of iterations $K$ and number of executions of each event type $N^{\xi}$.

### *6.4. Mathematical programming model*

The MPR represents the dynamics of the simulated system, equivalent to the event-scheduling algorithm, i.e., Algorithm 3. Specifically, execution scheduling times, execution occurring times and state variable changes during the simulation run can all be seen in the MPR as decision variables. The scheduling times and the occurring times of the execution of event type $\xi$ indexed by $i$ are denoted by $e_i^{\xi,0}$ and $e_i^{\xi,1}$, respectively. Variables $\mathcal{E}_k$ denotes the simulation clock at the beginning of iteration $k$ in Algorithm 3. Variables $e_i^{\xi,0}$, $e_i^{\xi,1}$ and $\mathcal{E}_k$ are all real-valued and non-negative. Variable $u_k^s$ is used to denote the value of state variable $s$ at the beginning of iteration $k$ in Algorithm 3. Variables $u_k^s$ are integer according to assumption (1). Some binary variables are also used in the MPR, and they will be introduced in the following, during the explanation of the model.

The set of all events $\xi$ is denoted by $\mathbb{X}$. $\mathbb{I}^{\xi}$ denotes the set $\{1, ..., N^{\xi}\}$, which is the number of executions of event $\xi$, and $\mathbb{K}$ denotes the set $\{0, ..., K\}$, which is the total number of executions in the simulation run. $\mathbb{S}$ denotes the set of all state variables. $\mathbb{S}^{\xi}$ and $\mathbb{S}^{\bar{\xi}}$ denote the set of state variables relevant to scheduling and cancellation

**State variables**

| $s$ | Description | Initial value |
|---|---|---|
| $g$ | Number of occupied servers | 0 |
| $q$ | Number of jobs in the queue | 0 |
| $u^{arr}$ | Number of executions with event type $arr$ in the future event list | 0 |

**Zero-delay events**

| $\xi$ | Description | Condition to schedule | State transition |
|---|---|---|---|
| $a\tilde{r}r$ | Counting arrival | $u^{arr} \leq 0$ | $u^{arr}$++ |
| $ss$ | Start | $1 \leq q, g \leq m-1$ | $g$++,$q$-- |

**Positive-delay events**

| $\xi$ | Description | Delay | Counting event | Counting variable | State transition |
|---|---|---|---|---|---|
| $arr$ | Arrival | $T^{arr}$ | $a\tilde{r}r$ | $u^{arr}$ | $q$++,$u^{arr}$-- |
| $sf$ | Finish | $T^{sf}$ | $ss$ | $g$ | $g$-- |

**Table 2.** Events to simulate G/G/m system.

conditions of event type $\xi$, respectively. $\mathbb{S}^c$ denotes the set of counting variables.

### *6.4.1. Event execution*

The first group of mathematical relationships, denoted by group-A, are the constraints for performing one execution in one iteration of Algorithm 3. Binary variables $w_{i,k}^{\xi}$ are introduced to represent the $i$-th execution of event type $\xi$ occurring in iteration $k-1$. The $i$-th execution of event $\xi$ is performed in iteration $k-1$ if and only if $w_{i,k}^{\xi}$ is equal to one, and variable $e_{i,1}^{\xi}$ is binded to $\mathcal{E}_k$, as shown in constraints (A1) and (A2). Constraints (A3) state that in each iteration, one and only one execution occurs. Constraints (A4) state that an execution occurs at most once. Constraints (A5) imply that the simulation cannot be reversed from iteration $k-1$ to iteration $k$. Constraint (A6) implies that the simulation clock is initialized to zero. Constraints (A7) implies that the delay between the scheduling and occurrence of an execution is equal to a sample from the random variate $T^{\xi}$.

$$e_i^{\xi,1} - \mathcal{E}_k \geq M(w_{i,k}^{\xi} - 1) \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \quad (A1)$$

$$\mathcal{E}_k - e_i^{\xi,1} \geq M(w_{i,k}^{\xi} - 1) \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \quad (A2)$$

$$\sum_{k \in \mathbb{K}} w_{i,k}^{\xi} \leq 1 \qquad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi} \quad (A3)$$

$$\sum_{\xi \in \mathbb{X}} \sum_{i \in \mathbb{I}^{\xi}} w_{i,k}^{\xi} = 1 \qquad \forall\, k \in \mathbb{K} \quad (A4)$$

$$\mathcal{E}_k - \mathcal{E}_{k-1} \geq 0 \qquad \forall\, k \in \mathbb{K} \quad (A5)$$

$$\mathcal{E}_0 = 0 \quad (A6)$$

$$e_i^{\xi,1} - e_i^{\xi,0} = t_i^{\xi} \qquad \forall \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi} \quad (A7)$$

*6.4.2. Constraints for scheduling new events*

The second group of constraints, denoted by group B, state the scheduling of new executions. Binary variables $x^{\xi}_{i,k}$ are used to represent that an execution with event type $\xi$ and index $i$ is scheduled in iteration $k$ if it is equal to one. Constraints (B1) and (B2) shows that the scheduling time of the execution is equal to the simulation clock when it is scheduled.

$$e^{\xi,0}_i - \mathcal{E}_k \geq M(x^{\xi}_{i,k} - 1) \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K}, i \in \mathbb{I}^{\xi} \quad (B1)$$
$$\mathcal{E}_k - e^{\xi,0}_i \geq M(x^{\xi}_{i,k} - 1) \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K}, i \in \mathbb{I}^{\xi} \quad (B2)$$

To schedule zero-delay events, the condition to schedule must be verified, as stated in lines 2 to 8 in Algorithm 3, and it is also shown by constraints (B3) to (B11). Binary variables $z^{\xi}_k$ represents that the system state satisfies the condition to schedule event $\xi$ and all the previously scheduled executions have been performed, i.e., there is no execution of $\xi$ in the future event list. In this case, it is mandatory to schedule an event $\xi$ in iteration $k$ in Algorithm 3, if $z^{\xi}_k$ is equal to one. Constraints (B3) and (B4) imply that if $z^{\xi}_k$ is equal to one, the state variables satisfy the condition to schedule execution of event type $\xi$. Moreover, a set of binary variables $v^{\xi,s,0}_k$ and $v^{\xi,s,1}_k$ are used to verify if the condition to schedule $\xi$ is false. Specifically, constraints (B5) state that if $v^{\xi,s,0}_k$ is equal to one, $u^s_k$ will be smaller than $a^{\xi,s}$, and hence, the inequality $a^{\xi,s} \leq s$ is violated. Similar to constraints (B6), if $v^{\xi,s,1}_k$ is equal to one, $s \leq c^{\xi,s}$ is violated.

$$u^s_k - a^{\xi,s} \geq M(z^{\xi}_k - 1) \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K}, s \in \mathbb{S}^{\xi},\ T^{\xi} = 0 \quad (B3)$$
$$c^{\xi,s} - u^s_k \geq M(z^{\xi}_k - 1) \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K}, s \in \mathbb{S}^{\xi},\ T^{\xi} = 0 \quad (B4)$$
$$(a^{\xi,s} - 1) - u^s_k \geq M(v^{\xi,s,0}_k - 1) \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K}, s \in \mathbb{S}^{\xi},\ T^{\xi} = 0 \quad (B5)$$
$$u^s_k - (c^{\xi,s} + 1) \geq M(v^{\xi,s,1}_k - 1) \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K}, s \in \mathbb{S}^{\xi},\ T^{\xi} = 0 \quad (B6)$$

Binary variable $u^{\xi}_k$ is introduced to verify if there is an execution of event type $\xi$ in the future event list at the beginning of iteration $k$. $u^{\xi}_0$ is initialized to zero as in constraints (B8), showing that there is no execution in the future event list. Constraints (B7) shows that $u^{\xi}_k$ is turned to one if an execution of event type $\xi$ is scheduled and turned to zero if an execution of event type $\xi$ is performed in iteration $(k-1)$. Constraints (B9) specify that $z^{\xi}_k$ is equal to zero only if at least one of the above mentioned conditions is violated, i.e., either the condition to schedule is violated, or there is already an execution in the future event list.

$$u^{\xi}_k = u^{\xi}_{k-1} - \sum_{i \in \mathbb{I}^{\xi}} w^{\xi}_{i,k} + z^{\xi}_{k-1} \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K},\ T^{\xi} = 0 \quad (B7)$$
$$u^{\xi}_0 = 0 \qquad \forall\, \xi \in \mathbb{X},\ T^{\xi} = 0 \qquad (B8)$$
$$1 - z^{\xi}_k \leq \sum_{s \in \mathbb{S}^{\xi}} v^{\xi,s,0}_k + \sum_{s \in \mathbb{S}^{\xi}} v^{\xi,s,1}_k + u^{\xi}_k \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K},\ T^{\xi} = 0 \quad (B9)$$

Constraints (B10) show that if $z^{\xi}_k$ is equal to one, one execution of event type $\xi$ is scheduled in iteration $k$. Constraints (B11) state that executions of event type $\xi$ have

different indices.

$$\sum_{i\in\mathbb{I}^{\xi}} x_{i,k}^{\xi} = z_k^{\xi} \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K},\ T^{\xi} = 0 \quad (B10)$$

$$\sum_{k\in\mathbb{K}} x_{i,k}^{\xi} \leq 1 \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi},\ T^{\xi} = 0 \quad (B11)$$

To schedule positive-delay events, an execution of the counting event must be performed. Constraints (B12) show that one execution of positive-delay event $\xi$ is scheduled after each execution of its counting event $\tilde{\xi}$.

$$x_{i,k}^{\xi} = w_{i,k}^{\tilde{\xi}} \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi}, T^{\xi} > 0 \quad (B12)$$

*6.4.3. Constraints for sequencing executions*

The index $i$ of executions represents the scheduling sequence, i.e., if an execution is scheduled in an earlier iteration, then its index will be smaller. Moreover, an execution must by performed after being scheduled. Group-C constraints force such sequences, which are relevant to all events. Constraints (C1) show that if the $i$-th execution of event $\xi$ is not scheduled before simulation termination, then the $(i+1)$-th execution will not be scheduled. Constraints (C2) state that if the $i$-th execution of event $\xi$ is not scheduled before simulation termination, then it will not be performed. Constraints (C3) state that an execution cannot be performed before being scheduled, unless it remains in the future event list at the end of the simulation run. Constraints (C4) depict that the $(i+1)$-th execution of event $\xi$ must be scheduled after the $i$-th execution, unless the $(i+1)$-th execution is not scheduled before simulation termination.

$$\sum_{k\in\mathbb{K}} x_{i+1,k}^{\xi} - \sum_{k\in\mathbb{K}} x_{i,k}^{\xi} \leq 0 \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi} \quad (C1)$$

$$\sum_{k\in\mathbb{K}} w_{i,k}^{\xi} - \sum_{k\in\mathbb{K}} x_{i,k}^{\xi} \leq 0 \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi} \quad (C2)$$

$$\sum_{k\in\mathbb{K}} k w_{i,k}^{\xi} - \sum_{k\in\mathbb{K}} k x_{i,k}^{\xi} \geq 1 + M(\sum_{k\in\mathbb{K}} w_{i,k}^{\xi} - 1) \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi} \quad (C3)$$

$$\sum_{k\in\mathbb{K}} k x_{i+1,k}^{\xi} - \sum_{k\in\mathbb{K}} k x_{i,k}^{\xi} \geq 1 + M(\sum_{k\in\mathbb{K}} x_{i+1,k}^{\xi} - 1) \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi} \quad (C4)$$

*6.4.4. Constraints for event cancellation*

The fourth group of constraints, denoted by group-D, state that executions of event type $\xi$ in the future event list are canceled if the cancellation condition is true. Similar to constraints (B3) to (B6), constraint (D1) to (D4) show that if binary variables $z_k^{\bar{\xi}}$ are equal to one, then the cancellation condition of event $\xi$ is true at the beginning of iteration $k$, where binary variables $z_k^{\bar{\xi}}$, $v_k^{\bar{\xi},s,0}$ and $v_k^{\bar{\xi},s,1}$ are the counter parts of $z_k^{\xi}$, $v_k^{\xi,s,0}$ and $v_k^{\xi,s,1}$, but for event cancellation rather than event scheduling. Constraints

(D5) show that $z_k^{\bar{\xi}}$ can be equal to zero only if the cancellation condition is false.

$$u_k^s - a^{\bar{\xi},s} \geq M(z_k^{\bar{\xi}} - 1) \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K}, s \in \mathbb{S}^{\bar{\xi}} \quad (D1)$$

$$c^{\bar{\xi},s} - u_k^s \geq M(z_k^{\bar{\xi}} - 1) \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K}, s \in \mathbb{S}^{\bar{\xi}} \quad (D2)$$

$$(a^{\bar{\xi},s} - 1) - u_k^s \geq M(v_k^{\bar{\xi},s,0} - 1) \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K}, s \in \mathbb{S}^{\bar{\xi}} \quad (D3)$$

$$u_k^s - (c^{\bar{\xi},s} + 1) \geq M(v_k^{\bar{\xi},s,1} - 1) \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K}, s \in \mathbb{S}^{\bar{\xi}} \quad (D4)$$

$$1 - z_k^{\bar{\xi}} \leq \sum_{s \in \mathbb{S}^{\bar{\xi}}} v_k^{\bar{\xi},s,0} + \sum_{s \in \mathbb{S}^{\bar{\xi}}} v_k^{\bar{\xi},s,1} \quad \forall\, \xi \in \mathbb{X}, k \in \mathbb{K} \quad (D5)$$

The $i$-th execution of event type $\xi$ is canceled if there exists an iteration $k$ where the cancellation condition is true, and it is scheduled before and performed after iteration $k$. The $i$-th execution of event type $\xi$ is scheduled in iteration $k_i^{\xi,0}$ and performed in iteration $(k_i^{\xi,1} - 1)$, where both $k_i^{\xi,0}$ and $k_i^{\xi,1}$ are integer variables. The value of $k_i^{\xi,0}$ and $k_i^{\xi,1}$ are calculated as in constraints (D6) and (D7). The $i$-th execution of event type $\xi$ will be canceled if there exists $k$ between $k_i^{\xi,0}$ and $k_i^{\xi,1}$ such that $z_k^{\bar{\xi}}$ is equal to one. Binary variables $\theta_{i,k}^{\xi}$ equal to one are used to represent the existence of such a $k$, which is guaranteed by constraints (D8) and (D9).

$$k_i^{\xi,1} = \sum_{k \in \mathbb{K}} k w_{k,i}^{\xi} \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi} \quad (D6)$$

$$k_i^{\xi,0} = \sum_{k \in \mathbb{K}} k x_{k,i}^{\xi} \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi} \quad (D7)$$

$$k z_k^{\bar{\xi}} - k_i^{\xi,0} - 1 \geq M(\theta_{i,k}^{\xi} - 1) \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \quad (D8)$$

$$k_i^{\xi,1} - 1 - k z_k^{\bar{\xi}} \geq M(\theta_{i,k}^{\xi} - 1) \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \quad (D9)$$

Also, binary variables $\phi_{i,k}^{\xi,0}$ and $\phi_{i,k}^{\xi,1}$ and constraints (D12) are used to avoid not canceling the executions when the condition to cancellation is true. Specifically, variables $\phi_{i,k}^{\xi,0}$ and $\phi_{i,k}^{\xi,1}$ equal to one represent if the condition to cancel is true in iteration $k$ but it is earlier than the scheduling of $e_i^{\xi}$ or later than performing $e_i^{\xi}$, respectively, as stated by constraints (D10) and (D11). Constraints (D12) state that $\theta_{i,k}^{\xi}$ is equal to zero only if at least one between $\phi_{i,k}^{\xi,0}$ and $\phi_{i,k}^{\xi,1}$ is equal to one.

$$k_i^{\xi,0} - k z_k^{\bar{\xi}} \geq M(\phi_{i,k}^{\xi,0} - 1) \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \quad (D10)$$

$$k z_k^{\bar{\xi}} - k_i^{\xi,1} \geq M(\phi_{i,k}^{\xi,1} - 1) \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \quad (D11)$$

$$1 - \theta_{i,k}^{\xi} \leq \phi_{i,k}^{\xi,0} + \phi_{i,k}^{\xi,1} \quad \forall\, \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \quad (D12)$$

As introduced in section 2.4.1, binary variable $w_{i,k}^{\xi}$ equal to one represents that the $i$-th execution of event $\xi$ is performed in iteration $k$. Binary variable $\gamma_{i,k}^{\xi}$ is now introduced to indicate if that execution is canceled. $\gamma_{i,k}^{\xi}$ equal to one indicates that the $i$-th execution of event $\xi$ is performed in iteration $(k-1)$ and the state transition

function is applied. Specifically, if $w^{\xi}_{i,k}$ is equal to one, and it is not canceled, i.e., $\sum_{k' \in \mathbb{K}} \theta^{\xi}_{i,k'}$ is equal to zero, then $\gamma^{\xi}_{i,k}$ is equal to one. Otherwise, if $w^{\xi}_{i,k}$ is equal to zero, or $\sum_{k' \in \mathbb{K}} \theta^{\xi}_{i,k'}$ is greater than one, that execution is canceled, and $\gamma^{\xi}_{i,k}$ is equal to zero. Those relationships are stated by constraints (D13) and (D14).

$$\gamma^{\xi}_{i,k} \geq w^{\xi}_{i,k} - \sum_{k' \in \mathbb{K}} \theta^{\xi}_{i,k'} \quad \forall\ \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \quad (D13)$$

$$w^{\xi}_{i,k} - \sum_{k' \in \mathbb{K}} \theta^{\xi}_{i,k'} - 1 \geq M(\gamma^{\xi}_{i,k} - 1) \quad \forall\ \xi \in \mathbb{X}, i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \quad (D14)$$

*6.4.5. Constraints for state evolution*

The value of state variables $u^s$ are changed from $u^s_k$ to $u^s_{k+1}$ in iteration $k$, as stated in lines 22 and 25 in Algorithm 3. Constraints (E1) represent the evolution of the state variables if it is not a counting variable. Specifically, if an execution of event type $\xi$ is performed and not canceled in iteration $k$, the state variable $s$ is changed by function $F^{\xi,s}(u^s_k)$. Constraints (E2) to (E4) show the evolution of the counting variable $u^{\xi}$ of positive-delay events $\xi$. If $z^{\bar{\xi}}_k$, representing event cancellation, is equal to one, i.e., executions of event $\xi$ are canceled in iteration $k$, $u^{\xi}$ is set to zero, as in (E2). Otherwise, it is increased by one if a counting event $\tilde{\xi}$ is executed and decreased by one if event $\xi$ itself is executed.

$$u^s_{k+1} = \sum_{\xi \in \mathbb{X}} \sum_{i \in \mathbb{I}^{\xi}} \gamma^{\xi}_{i,k+1} F^{\xi,s}(u^s_k) \qquad \forall\ s \in \mathbb{S}/\mathbb{S}^c, k \in \mathbb{K} \qquad (E1)$$

$$u^{\xi}_{k+1} \leq M(1 - z^{\bar{\xi}}_k) \quad \forall\ \xi \in \mathbb{X}, t^{\xi} > 0, k \in \mathbb{K} \quad (E2)$$

$$u^{\xi}_{k+1} \leq u^{\xi}_k - \sum_{i \in \mathbb{I}^{\xi}} \gamma^{\xi}_{i,k+1} + \sum_{i \in \mathbb{I}^{\xi}} \gamma^{\tilde{\xi}}_{i,k+1} + M z^{\bar{\xi}}_k \quad \forall\ \xi \in \mathbb{X}, t^{\xi} > 0, k \in \mathbb{K} \quad (E3)$$

$$u^{\xi}_{k+1} \geq u^{\xi}_k - \sum_{i \in \mathbb{I}^{\xi}} \gamma^{\xi}_{i,k+1} + \sum_{i \in \mathbb{I}^{\xi}} \gamma^{\tilde{\xi}}_{i,k+1} - M z^{\bar{\xi}}_k \quad \forall\ \xi \in \mathbb{X}, t^{\xi} > 0, k \in \mathbb{K} \quad (E4)$$

If all the events change the state variables with a fixed increment or decrement equal to $\Delta^{\xi,s}$, constraints (E1) will be changed to (E5), which are linear constraints, and the MPR becomes a MILP.

$$u^s_{k+1} = u^s_k + \sum_{\xi \in \mathbb{X}} \sum_{i \in \mathbb{I}^{\xi}} \gamma^{\xi}_{i,k+1} \Delta^{\xi,s} \quad \forall\ s \in \mathbb{S}, k \in \mathbb{K} \quad (E5)$$

If event $\xi$ cannot be canceled, variables $\gamma^{\xi}_{i,k}$ and group-D constraints are not introduced, and the variables $\gamma^{\xi}_{i,k}$ are replaced by $w^{\xi}_{i,k}$ in the constraints (E1) and (E5). Constraints (E2) to (E4) are replaced by (E6).

$$u^{\xi}_{k+1} = u^{\xi}_k - \sum_{i \in \mathbb{I}^{\xi}} w^{\xi}_{i,k+1} + \sum_{i \in \mathbb{I}^{\xi}} w^{\tilde{\xi}}_{i,k+1} \quad \forall\ k \in \mathbb{K} \quad (E6)$$

The initial value of each state variable $s$ should be given, denoted by $s_0$, which is

shown with constraints (E7).

$$u_0^s = s_0 \quad \forall\ s \in \mathbb{S} \quad (E7)$$

*6.4.6. Objective function*

With the constraints defined in the previous sections, there is a unique feasible solution in terms of execution scheduling times, execution occurring times, history of simulation clock, i.e., $e_i^{\xi,0}$, $e_i^{\xi,1}$ and $\mathcal{E}_k$. Thus, the objective function can be any function of those variables, for instance, average of system time or waiting time in queueing systems. Multiple feasible solutions may appear in terms of binary variables, since the sequence of executions with identical execution time is not uniquely defined. However, this will not impact event execution time thanks to assumption (4).

The flexibility of the objective function definition is a main difference between the formulation proposed by Chan and Schruben (2008) and this work, since the objective function of MPR in Chan and Schruben (2008) can be only the sum of all execution times. The uniqueness of optimal solution and flexibility of objective function formulation is particularly beneficial if one wants to make use of the resulting MPR to solve an optimization problem concerning the design or operation of the discrete event system (for instance the capacity of the queue or the control policy of a manufacturing system), since changing the objective function or adding new constraints to calculate the system performance will not influence the equivalence of MPR and a simulation run. This issue will be discussed in Section 4.

## 7. Applications

In this section, several examples are presented, including a G/G/m queue, a merge queueing system composed by three single server stations, and a single server queue with failure as an example of event cancellation. The equivalence of MPR solution and history of a simulation run has been validated with $K$ equal to 20 for 100 independent replicates.

### *7.1. G/G/m queue*

The first example is a G/G/m queue. Table 2 shows the state variables and events composing the DES model, and the detailed explanation of the state variables and events can be found in section 6.3. The MPR generated by the approach proposed in this work is as follows.

Group-A, group-C and (B1) (B2) constraints are identical for all systems and all event type as presented in section 6.4.1, 6.4.2 and 6.4.3.

$$(A1) - (A7), (B1), (B2), (C1) - (C4) \qquad \forall \xi \in \{arr, a\tilde{r}r, ss, sf\}$$

Events $a\tilde{r}r$ and $ss$ are zero-delay, so constraints (B3) to (B11) are applied. Events $arr$ and $sf$ are positive-delay, so constraints (B13) are applied. Specifically, constraints (B7), (B8), (B10) and (B11) are identical for all systems, and hence, for sake of simplicity, they are not expanded.

$$(B7), (B8), (B10), (B11) \qquad \forall \xi \in \{a\tilde{r}r, ss\}$$

Constraints (1) are constraints (B4) of the counting event of arrival, stating that if a counting event of arrival is scheduled, i.e., $z_k^{a\tilde{r}r}$ is equal to one, there must be no execution of arrival in the future event list. Constraints (2) imply that if $v_k^{a\tilde{r}r,arr,1}$ is equal to one, there must be no execution of event $arr$ in the future event list, hence, at most one between $z_k^{a\tilde{r}r}$ and $v_k^{a\tilde{r}r,arr,1}$ can be equal to one. Constraints (3) are constraints (B9) of event $a\tilde{r}r$, and imply that if event $a\tilde{r}r$ is not scheduled, it is either because there is already an execution of $arr$ or an execution of $a\tilde{r}r$ in the future event list.

$$1 - u_k^{arr} \geq z_k^{a\tilde{r}r} \qquad \forall k \in \mathbb{K} \tag{1}$$

$$u_k^{arr} \geq v_k^{a\tilde{r}r,arr,1} \qquad \forall k \in \mathbb{K} \tag{2}$$

$$1 - z_k^{a\tilde{r}r} \leq v_k^{a\tilde{r}r,arr,1} + u_k^{a\tilde{r}r} \qquad \forall k \in \mathbb{K} \tag{3}$$

Constraints (4) are constraints (B4) of event $ss$ concerning state variable $q$, stating that if an execution of $ss$ is scheduled, i.e., $z_k^{ss}$ is equal to one, there must be at least one job waiting in the queue. Constraints (5) imply that if $v_k^{ss,q,0}$ is equal to one, there must be no job waiting in the queue, hence, at most one between $z_k^{ss}$ and $v_k^{ss,q,0}$ can be equal to one. Constraints (6) are constraints (B4) of event $ss$ concerning state variable $g$, stating that if an execution of $ss$ is scheduled, there must be at least one server available. Constraints (7) imply that if $v_k^{ss,g,1}$ is equal to one, all the servers are occupied, hence, at most one between $z_k^{ss}$ and $v_k^{ss,g,1}$ can be equal to one. Constraints (8) are constraints (B9) of event $ss$, and imply that if event $ss$ is not scheduled, it is either because there is no job in the queue, or no server is available, or there is already an execution of $ss$ in the future event list.

$$u_k^q \geq z_k^{ss} \qquad \forall k \in \mathbb{K} \tag{4}$$

$$-u_k^q \geq K(v_k^{ss,q,0} - 1) \qquad \forall k \in \mathbb{K} \tag{5}$$

$$m - u_k^g \geq z_k^{ss} \qquad \forall k \in \mathbb{K} \tag{6}$$

$$u_k^g \geq m v_k^{ss,g,1} \qquad \forall k \in \mathbb{K} \tag{7}$$

$$1 - z_k^{ss} \leq v_k^{ss,q,0} + v_k^{ss,g,1} + u_k^{ss} \qquad \forall k \in \mathbb{K} \tag{8}$$

Since arrival event $arr$ and finish event $sf$ are positive-delay, constraints (B12) should be applied, as it can be seen in constraints (9) and (10). An execution of arrival event is scheduled if a counting event $a\tilde{r}r$ is executed. An execution of finish event is scheduled if a counting event $ss$ is executed.

$$x_{i,k}^{arr} = w_{i,k}^{a\tilde{r}r} \qquad \forall i \in \mathbb{I}, k \in \mathbb{K} \tag{9}$$

$$x_{i,k}^{sf} = w_{i,k}^{ss} \qquad \forall i \in \mathbb{I}, k \in \mathbb{K} \tag{10}$$

Constraints (11) show the evolution of state variable $u^{arr}$, i.e., the number of execution of $arr$ in the future event list. It is incremented by one if the counting event $a\tilde{r}r$ is executed, and decremented by one if the arrival event $arr$ is executed, i.e., a job arrives. Constraints (12) show the evolution of state variable $q$, i.e., the queue level. It is incremented by one if a job arrives, and decremented by one if a job starts to be served. Constraints (13) show the evolution of state variable $g$, i.e., number of occupied servers. It is incremented by one if a job starts to be served, and decremented by one

if a job is released. The rest of the model indicates the initialization and range of state variables.

$$u_k^{arr} = u_{k-1}^{arr} - \sum_{i \in \mathbb{I}} w_{i,k}^{arr} + \sum_{i \in \mathbb{I}} w_{i,k}^{\tilde{arr}} \qquad \forall k \in \mathbb{K} \tag{11}$$

$$u_k^{q} = u_{k-1}^{q} - \sum_{i \in \mathbb{I}} w_{i,k}^{ss} + \sum_{i \in \mathbb{I}} w_{i,k}^{arr} \qquad \forall k \in \mathbb{K} \tag{12}$$

$$u_k^{g} = u_{k-1}^{g} - \sum_{i \in \mathbb{I}} w_{i,k}^{fs} + \sum_{i \in \mathbb{I}} w_{i,k}^{ss} \qquad \forall k \in \mathbb{K} \tag{13}$$

$$u_0^{arr} = u_0^{q} = u_0^{g} = 0$$

$$u_k^{arr} \in \{0,1\},\ u_k^{g} \in \{0,...,m\},\ u_k^{q} \in \mathbb{N}$$

Table 3 shows a simulation run of a G/G/m queue including 10 executions. In iteration zero, the system is in initial state, and this state satisfies the condition to schedule the counting event of arrival, as in lines 2 to 8 in Algorithm 3. In the solution of MPR, the scheduling of $a\tilde{r}r$ is represented by $z_0^{a\tilde{r}r}$ and $x_{1,0}^{a\tilde{r}r}$ equal to 1, and its occurring time is equal to the simulation clock, which is equal to 0. The execution $(a\tilde{r}r, 1, e_1^{a\tilde{r}r,0}, e_1^{a\tilde{r}r,1})$, denoted by $e_1^{a\tilde{r}r}$ in the table, is then performed since it is the earliest execution (line 18 in Algorithm 3), which is also indicated by $w_{1,1}^{a\tilde{r}r}$ equal to one, and simulation clock in the next iteration, i.e., iteration 1, is equal to the occurring time $e_{1,1}^{a\tilde{r}r}$ in the solution of MPR. After that, the positive-delay event $arr$ is then scheduled (lines 26 to 30 in Algorithm 3), which can also be seen from the MPR solution, since $x_{1,1}^{arr}$ is equal to one, the scheduling time of $e_{1,0}^{arr}$ is equal to the current simulation clock time and the occurrence time is 2.3 time unit later than the scheduling time, where 2.3 is sampled from the distribution of inter-arrival time $T^{arr}$. At the end of the iteration, the system state is updated according to the state transition function of event $a\tilde{r}r$ (line 25 in Algorithm 3), which in the MPR solution is indicated by an increment equal to one of state variable $u^{arr}$. This procedure is repeated in the next iterations, and the equivalence of DES procedure and the solution of MPR can be seen from Table 3.

**Table 3.** Simulation run and MPR solution of G/G/m queue.

| $k$ | *clock* | New zero-delay executions | Future event list | Perform execution | New positive-delay execution | System state |
|---|---|---|---|---|---|---|
| 0 | 0 | $e_1^{a\tilde{r}r} = (a\tilde{r}r, 1, 0, 0)$ | $\{e_1^{a\tilde{r}r}\}$ | $e_1^{a\tilde{r}r}$ | $e_1^{arr} = (arr, 1, 0, 2.3)$ | $(u^{arr}, q, g) = (1, 0, 0)$ |
| | $\mathcal{E}_0 = 0$ | $z_0^{a\tilde{r}r} = 1, x_{1,0}^{a\tilde{r}r} = 1, e_{1,1}^{a\tilde{r}r} = 0$ | | $w_{1,1}^{a\tilde{r}r} = 1, \mathcal{E}_1 = e_{1,1}^{a\tilde{r}r}$ | $x_{1,1}^{arr} = 1, e_{1,1}^{arr} = 0, e_{1,1}^{arr} = 2.3$ | $u_1^{arr} = 1, u_1^q = 0, u_1^g = 0$ |
| 1 | 0 | | $\{e_1^{arr}\}$ | $e_1^{arr}$ | | $(u^{arr}, q, g) = (0, 1, 0)$ |
| | $\mathcal{E}_1 = 0$ | | | $w_{1,2}^{arr} = 1, \mathcal{E}_2 = e_{1,1}^{arr}$ | | $u_2^{arr} = 0, u_2^q = 1, u_2^g = 0$ |
| 2 | 2.3 | $e_2^{a\tilde{r}r} = (a\tilde{r}r, 2, 2.3, 2.3)$<br>$e_1^{ss} = (ss, 1, 2.3, 2.3)$ | $\{e_2^{a\tilde{r}r}, e_1^{ss}\}$ | $e_2^{a\tilde{r}r}$ | $e_2^{arr} = (arr, 2, 2.3, 11.1)$ | $(u^{arr}, q, g) = (1, 1, 0)$ |
| | $\mathcal{E}_2 = 2.3$ | $z_2^{a\tilde{r}r} = 1, x_{2,2}^{a\tilde{r}r} = 1, e_{2,1}^{a\tilde{r}r} = 2.3$<br>$z_2^{ss} = 1, x_{1,2}^{ss} = 1, e_{1,1}^{ss} = 2.3$ | | $w_{2,3}^{a\tilde{r}r} = 1, \mathcal{E}_3 = e_{2,1}^{a\tilde{r}r}$ | $x_{2,3}^{arr} = 1, e_{2,0}^{arr} = 2.3, e_{2,1}^{arr} = 11.1$ | $u_3^{arr} = 1, u_3^q = 1, u_3^g = 0$ |
| 3 | 2.3 | | $\{e_1^{ss}, e_2^{arr}\}$ | $e_1^{ss}$ | $e_1^{sf} = (sf, 1, 2.3, 6.0)$ | $(u^{arr}, q, g) = (1, 0, 1)$ |
| | $\mathcal{E}_3 = 2.3$ | | | $w_{1,4}^{ss} = 1, \mathcal{E}_4 = e_{1,1}^{ss}$ | $x_{1,4}^{sf} = 1, e_{1,0}^{sf} = 2.3, e_{1,1}^{sf} = 6.0$ | $u_4^{arr} = 1, u_4^q = 0, u_4^g = 1$ |
| 4 | 2.3 | | $\{e_1^{sf}, e_2^{arr}\}$ | $e_1^{sf}$ | | $(u^{arr}, q, g) = (1, 0, 0)$ |
| | $\mathcal{E}_4 = 2.3$ | | | $w_{1,5}^{sf} = 1, \mathcal{E}_5 = e_{1,1}^{sf}$ | | $u_5^{arr} = 1, u_5^q = 0, u_5^g = 0$ |
| 5 | 6.0 | | $\{e_2^{arr}\}$ | $e_2^{arr}$ | | $(u^{arr}, q, g) = (0, 1, 0)$ |
| | $\mathcal{E}_5 = 2.3$ | | | $w_{2,6}^{arr} = 1, \mathcal{E}_6 = e_{2,1}^{arr}$ | | $u_6^{arr} = 0, u_6^q = 1, u_6^g = 0$ |
| 6 | 11.1 | $e_3^{a\tilde{r}r} = (a\tilde{r}r, 3, 11.1, 11.1)$<br>$e_2^{ss} = (ss, 2, 11.1, 11.1)$ | $\{e_3^{a\tilde{r}r}, e_2^{ss}\}$ | $e_3^{a\tilde{r}r}$ | $e_3^{arr} = (arr, 3, 11.1, 12.1)$ | $(u^{arr}, q, g) = (1, 1, 0)$ |
| | $\mathcal{E}_6 = 11.1$ | $z_6^{a\tilde{r}r} = 1, x_{3,6}^{a\tilde{r}r} = 1, e_{3,1}^{a\tilde{r}r} = 11.1$<br>$z_6^{ss} = 1, x_{2,6}^{ss} = 1, e_{2,1}^{ss} = 11.1$ | | $w_{3,7}^{a\tilde{r}r} = 1, \mathcal{E}_7 = e_{3,1}^{a\tilde{r}r}$ | $x_{3,7}^{arr} = 1, e_{3,0}^{arr} = 11.1, e_{3,1}^{arr} = 12.1$ | $u_7^{arr} = 1, u_7^q = 1, u_7^g = 0$ |

### *7.2. Comparison with Schrunben models of G/G/m systems*

### *7.3. Optimizing server number of G/G/m queue*

### *7.4. Complex case: single-server merge*

A queueing system composed of three servers in a merge architecture (Figure 5) is presented in this section. Jobs enter the system at server 1 or server 2, and the buffers in front of the two servers have infinite capacity. It is assumed that all the jobs arrive at time zero. After processing a job, server 1 or 2 can release the job to the buffer with capacity $Q$ in front of server 3, denoted as buffer 3, if the buffer is not full. The blocking policy is blocking-after-service. If there is only one space available in the buffer, and both servers 1 and 2 are holding a finished job, server 1 will release the job. Server 3 can take jobs from buffer 3 and release the job immediately after the process is finished.

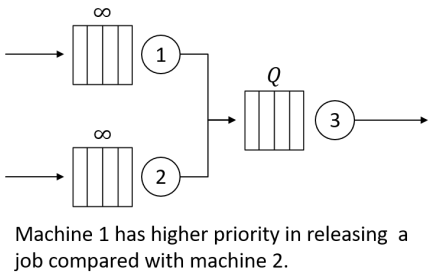


**Figure 5.** Example: single server merge.

State variables have to be defined in the first place, as in Table 4. Binary variable $g^j$ for $j = 1, 2, 3$ is equal to one if server $j$ is processing a job, otherwise it is equal to zero. Binary variable $b^j$ for $j = 1, 2$ is equal to one if server $j$ is holding a finished job, otherwise it is equal to zero. State variable $b^3$ is not defined, since the job will be released immediately after its processing. Integer variable $q$ represents the number of jobs waiting in buffer 3.

The events composing the DES model are then defined as in Table 4. Event $(ss, j)$ for $j = 1, 2, 3$ represents the starting of service of server $j$. If server $j$ is idle, i.e., neither working nor holding a finished part, $(ss, j)$ will be scheduled to execute immediately. After an $(ss, j)$ is executed, server $j$ becomes active, i.e., state variable $g^j$ is incremented by one. Besides, if server 3 starts a job, the buffer level $q$ is decreased by one. Event $(d, j)$ for $j = 1, 2$ represents the departure of a job from server $j$. A $(d, 1)$ can be executed if server 1 is holding a finished job and there is at least one space available in buffer 3. A $(d, 2)$ can be executed immediately if server 2 is holding a finished job, server 1 is not holding a job and there is at least one space available in buffer 3. After executing $(d, j)$, the server is not holding any job, hence, state variable $b^j$ becomes zero, and the buffer level $q$ is increased by one. Both $(ss, j)$ and $(d, j)$ can be executed immediately once scheduled, so they are zero-delay. Event $(sf, j)$ represents that server $j$ finishes a job, and it is scheduled immediately after $(ss, j)$ is executed, and the occurrence delay is equal to the service time, i.e., sampled from $T^j$. Thus, $(ss, j)$ is a counting event of $(sf, j)$. After event $(sf, j)$ is executed, server $j$ is no longer working, thus, $g^j$ is decreased by one. Thus, state variable $g^j$ is a counting variable of $(sf, j)$, since its value is incremented by one if $(ss, j)$ is executed, and decremented by one if $(sf, j)$ is executed. Specially for $j = 1, 2$, after a job is finished, the server holds a

finished job, and state variable $b^j$ is increased by one.

**State variables**

| $s$ | Description | Initial value |
|---|---|---|
| $g^j$ | Equal to 1 if server $j$ is working, otherwise equal to 0. | 0 |
| $b^j$ | Equal to 1 if server $j$ is blocked, otherwise equal to 0. | 0 |
| $q$ | Number of jobs in the queue. | 0 |

**Zero-delay events**

| $\xi$ | Description | Condition to schedule | State transition |
|---|---|---|---|
| $(ss, 1)$ | Start m1 | $g^1 \leq 0$ & $b^1 \leq 0$ | $g^1$++ |
| $(d, 1)$ | Depart m1 | $b^1 \geq 1$ & $q \leq Q - 1$ | $b^1$--, $q$++ |
| $(ss, 2)$ | Start m2 | $g^2 \leq 0$ & $b^2 \leq 0$ | $g^2$++ |
| $(d, 2)$ | Depart m2 | $b^2 \geq 1$ & $q \leq Q - 1$ & $b^1 \leq 0$ | $b^2$--, $q$++ |
| $(ss, 3)$ | Start m3 | $g^3 \leq 0$ & $q \geq 1$ | $g^3$++, $q$-- |

**Positive-delay events**

| $\xi$ | Description | Delay | Counting event | Counting variable | State transition |
|---|---|---|---|---|---|
| $(sf, 1)$ | Finish m1 | $T^{sf,1}$ | $(ss, 1)$ | $g^1$ | $g^1$--, $b^1$++ |
| $(sf, 2)$ | Finish m2 | $T^{sf,2}$ | $(ss, 2)$ | $g^2$ | $g^2$--, $b^2$++ |
| $(sf, 3)$ | Finish m3 | $T^{sf,3}$ | $(ss, 3)$ | $g^3$ | $g^3$-- |

**Table 4.** Events to simulate Single-server merge system.

The MPR proposed in this work is as follows.

Group-A, group-C and (B1) (B2) constraints are identical for all systems and all event type as presented in section 6.4.1, 6.4.2 and 6.4.3.

$$(A1) - (A7), (B1), (B2), (C1) - (C4) \qquad \forall \xi \in \{(ss, 1), (ss, 2), (ss, 3), (d, 1), (d, 2)\}$$
$$\forall \xi \in \{(sf, 1), (sf, 2), (sf, 3)\}$$

Events $(ss, j)$ for $j = 1, 2, 3$ and $(d, j)$ for $j = 1, 2$ are all zero-delay, constraints (B3) to (B11) should be applied. Specifically, constraints (B7), (B8), (B10) and (B11) are identical for all systems, and for sake of simplicity, they are not expanded.

$$(B7), (B8), (B10), (B11) \qquad \forall \xi \in \{(ss, 1), (ss, 2), (ss, 3), (d, 1), (d, 2)\}$$

Constraints (14) and (16), referring to constraints (B4), imply that if an execution of $(ss, j)$ is scheduled, machine $j$ is not occupied by a job under processing or finished. Constraints (15) and (17), referring to constraints (B6), imply that if variable $v_k^{ss,j,g^j,1}$ or $v_k^{ss,j,b^j,1}$ is equal to one, machine $j$ is occupied by a job under processing or a finished job. Constraints (18), referring to constraints (B9), state that if execution of event $(ss, 1)$ or $(ss, 2)$ is not scheduled, it is either because the machine is occupied,

or there is already an execution in the future event list.

$$-u_k^{g^j} \geq z_k^{ss,j} - 1 \qquad \forall\ j = 1,2, k \in \mathbb{K} \tag{14}$$

$$u_k^{g^j} \geq v_k^{ss,j,g^j,1} \qquad \forall\ j = 1,2, k \in \mathbb{K} \tag{15}$$

$$-u_k^{b^j} \geq z_k^{ss,j} - 1 \qquad \forall\ j = 1,2, k \in \mathbb{K} \tag{16}$$

$$u_k^{b^j} \geq v_k^{ss,j,b^j,1} \qquad \forall\ j = 1,2, k \in \mathbb{K} \tag{17}$$

$$1 - z_k^{ss,j} \leq v_k^{ss,j,g^j,1} + v_k^{ss,j,b^j,1} + u_k^{ss,j} \qquad \forall\ j = 1,2, k \in \mathbb{K} \tag{18}$$

Similarly, constraints (19) to (23) refer to constraints (B3) to (B6) and (B9) of event $(ss, 3)$, stating that an execution of event $(ss, 3)$ can be scheduled if and only if machine 3 is not occupied, there is a job waiting in buffer 3, there is no executions of $(ss, 3)$ in the future event list.

$$-u_k^{g^3} \geq z_k^{ss,3} - 1 \qquad \forall\ k \in \mathbb{K} \tag{19}$$

$$u_k^{g^3} \geq v_k^{ss,3,g^3,1} \qquad \forall\ k \in \mathbb{K} \tag{20}$$

$$u_k^{q} \geq z_k^{ss,3} \qquad \forall\ k \in \mathbb{K} \tag{21}$$

$$-u_k^{q} \geq Q(v_k^{ss,3,q,0} - 1) \qquad \forall\ k \in \mathbb{K} \tag{22}$$

$$1 - z_k^{ss,3} \leq v_k^{ss,3,g^3,1} + v_k^{ss,3,q,0} + u_k^{ss,3} \qquad \forall\ k \in \mathbb{K} \tag{23}$$

Constraints (24) to (28) refer to constraints (B3) to (B6) and (B9) of event $(d, 1)$, implying that an execution of event $(d, 1)$ can be scheduled if and only if there is a finished job in machine 1, there is space available in buffer 3, there is no executions of $(d, 1)$ in the future event list.

$$u_k^{b^1} \geq z_k^{d,1} \qquad \forall\ k \in \mathbb{K} \tag{24}$$

$$-u_k^{b^1} \geq v_k^{d,1,b^1,0} - 1 \qquad \forall\ k \in \mathbb{K} \tag{25}$$

$$Q - u_k^{q} \geq z_k^{d,1} \qquad \forall\ k \in \mathbb{K} \tag{26}$$

$$u_k^{q} \geq Q v_k^{d,1,q,1} \qquad \forall\ k \in \mathbb{K} \tag{27}$$

$$1 - z_k^{d,1} \leq v_k^{d,1,b^1,0} + v_k^{d,1,q,1} + u_k^{d,1} \qquad \forall\ k \in \mathbb{K} \tag{28}$$

Constraints (29) to (35) refer to constraints (B3) to (B6) and (B9) of event $(d, 2)$, depicting that an execution of event $(d, 2)$ can be scheduled if and only if there is a finished job in machine 2, there is space available in buffer 3, there is no finished job

waiting in machine 1, there is no executions of $(d, 2)$ in the future event list.

$$u_k^{b^2} \geq z_k^{d,2} \qquad \forall\ k \in \mathbb{K} \tag{29}$$

$$-u_k^{b^2} \geq v_k^{d,2,b^2,0} - 1 \qquad \forall\ k \in \mathbb{K} \tag{30}$$

$$Q - u_k^q \geq z_k^{d,2} \qquad \forall\ k \in \mathbb{K} \tag{31}$$

$$u_k^q \geq Q v_k^{d,2,q,1} \qquad \forall\ k \in \mathbb{K} \tag{32}$$

$$-u_k^{b^1} \geq z_k^{d,2} - 1 \qquad \forall\ k \in \mathbb{K} \tag{33}$$

$$u_k^{b^1} \geq v_k^{d,2,b^1,1} \qquad \forall\ k \in \mathbb{K} \tag{34}$$

$$1 - z_k^{d,2} \leq v_k^{d,2,b^2,0} + v_k^{d,2,q,1} + v_k^{d,2,b^1,1} + u_k^{d,2} \qquad \forall\ k \in \mathbb{K} \tag{35}$$

Since events $(sf, j)$ are positive-delay, constraints (B12) should be applied, as in constraints (36). A finishing event of machine $j$ can be scheduled after a starting event is executed.

$$x_{i,k}^{sf,j} = w_{i,k}^{ss,j} \qquad \forall\ j = 1, 2, 3, i \in \mathbb{I}^j, k \in \mathbb{K} \tag{36}$$

Constraints (37) to (39) indicate the evolution of state variables $g^j$, $b^j$ and $q$, respectively. $g^j$ is increased by one if machine $j$ starts a job and decreased by one if it finished a job. $b^j$ is increased by one if machine $j$ finishes a job, and decreased by one if it releases a job. $q$ is increased by one if machine 1 or 2 releases a job, and decreased by one if machine 3 seizes a job from buffer 3. The rest of the model indicates the initialization and range of state variables.

$$u_k^{g^j} = u_{k-1}^{g^j} + \sum_{i \in \mathbb{I}^j} w_{i,k}^{ss,j} - \sum_{i \in \mathbb{I}^j} w_{i,k}^{sf,j} \qquad \forall\ j = 1, 2, 3, k \in \mathbb{K} \tag{37}$$

$$u_k^{b^j} = u_{k-1}^{b^j} + \sum_{i \in \mathbb{I}^j} w_{i,k}^{sf,j} - \sum_{i \in \mathbb{I}^j} w_{i,k}^{d,j} \qquad \forall\ j = 1, 2, k \in \mathbb{K} \tag{38}$$

$$u_k^q = u_{k-1}^q + \sum_{j=1,2} \sum_{i \in \mathbb{I}^j} w_{i,k}^{d,j} - \sum_{i \in \mathbb{I}^3} w_{i,k}^{ss,3} \qquad \forall\ k \in \mathbb{K} \tag{39}$$

$$u_0^{g^1} = u_0^{g^2} = u_0^{g^3} = u_0^{b^1} = u_0^{b^2} = u_0^q = 0$$

$$u_k^{g^1}, u_k^{g^2}, u_k^{g^3}, u_k^{b^1}, u_k^{b^2} \in \{0, 1\},\ u_k^q \in \{0, ..., Q\} \qquad \forall\ k \in \mathbb{K}$$

### *7.5. G/G/1 with failure*

A G/G/1 queueing system with unreliable server is studied in this section. The server has two states, up and down, respectively. When the server is in up state, it can process a job. When the server turns to down state from up state, the repair starts immediately, and if the server is processing a job, the job is discarded. After the repair finishes, the server turns to up state, and restarts to process new jobs. The repair time follows a general distribution. The server will then keep in up state for a certain time called *up time*, following a general distribution, and will fail again. The failure is time-dependent. Jobs arrive at the system following a general arrival process and enter the queue in front of the server. When the server is in up state and idle, the server can take a job from the queue and start the service. After the service, a job can be immediately released from the system.

As in Table 5, the state variables include integer variable $q$ representing the number of jobs in the queue, binary variable $g$ indicating whether the server is working or not, binary variable $h$ indicating whether the server is in down state or not.

The events composing the DES model are then defined as in Table 5. Event $arr$ indicates that a job arrives at the system, and it is scheduled after a previous arrival, with a delay equal to the inter-arrival time. A counting event of $arr$, i.e., $\tilde{arr}$, and a counting variable $u^{arr}$ should be defined. The condition to schedule an arrival is that there is no execution of $arr$ in the future event list. When $\tilde{arr}$ is executed, the counting variable $u^{arr}$, i.e., the number of executions of $arr$ in the future event list, will be increased by one. When $arr$ is executed, the number of jobs in the queue will be increased by one, and the counting variable $u^{arr}$ will be decreased by one. Event $ss$ represents that the server starts to process a job. If and only if there is at least one job in the queue, the server is in up state and idle, event $ss$ is scheduled. When event $ss$ is executed, the number of jobs in the queue is reduced by one, and the server starts to work, i.e., state variable $g$ becomes one. Event $sf$ represents that a server finishes a job, and one execution of it is scheduled each time a job is started, and the occurrence delay is equal to the service time of the job. Event $ss$ is the counting event, and state variable $g$ is the counting variable of $sf$. After $sf$ is executed, the server becomes idle. Once the server is in down state, job under process is discarded, i.e., scheduled execution of $sf$ is canceled. Event $fl$ represents that the state of the server becomes down, and it is scheduled after the previous repair finishes with delay equal to the up time. Since it is positive-delay, a counting event $\tilde{fl}$ should be added, and also a counting variable $u^{fl}$. The condition to schedule an $\tilde{fl}$ is that the server is in up state, i.e., $h$ equal to zero, and there is no execution of $fl$ in the future event list, i.e., $u^{fl}$ smaller than or equal to zero. When $\tilde{fl}$ is executed, the number of executions of $fl$ is increased by one. When $fl$ is executed, the server is in down state, it cannot be working, i.e., $g$ equal to zero, and its number of executions in the future event list is reduced by one. Event $srp$ represents start of repair, and it is scheduled if the server is in down state and not under repair, and executed with no delay. When it is executed, the server is under repair. Event $frp$ states the finish of repair, and it is scheduled after event $srp$ is executed, hence, $srp$ is the counting event, and state variable $u^{frp}$ is introduced as the counting variable. After it is executed, the server becomes up, and no longer under repair.

**State variables**

| $s$ | Description | Initial value |
|---|---|---|
| $g$ | Equal to 1 if server is working on a job, otherwise equal to 0. | 0 |
| $h$ | Equal to 1 if server fails, otherwise equal to 0. | 0 |
| $q$ | Number of jobs in the queue. | 0 |
| $u^{frp}$ | Number of executions with event type $frp$ in the future event list. | 0 |
| $u^{arr}$ | Number of executions with event type $arr$ in the future event list. | 0 |
| $u^{fl}$ | Number of executions with event type $fl$ in the future event list. | 0 |

**Zero-delay events**

| $\xi$ | Description | Condition to schedule | State transition |
|---|---|---|---|
| $a\tilde{r}r$ | Counting arrival | $u^{arr} \leq 0$ | $u^{arr}$++ |
| $ss$ | Start | $1 \leq q,\ g \leq 0,\ h \leq 0$ | $g$++, $q$-- |
| $\tilde{fl}$ | Counting failure | $h \leq 0,\ u^{fl} \leq 0$ | $u^{fl}$++ |
| $srp$ | Start to repair | $h \geq 1,\ u^{frp} \leq 0$ | $u^{frp}$++ |

**Positive-delay events**

| $\xi$ | Description | Delay | Counting event | Counting variable | State transition | Condition to cancel |
|---|---|---|---|---|---|---|
| $arr$ | Arrival | $T^{arr}$ | $a\tilde{r}r$ | $u^{arr}$ | $u^{arr}$--, $q$++ | |
| $sf$ | Finish | $T^{sf}$ | $ss$ | $g$ | $g$-- | $h \geq 1$, $g \geq 1$ |
| $fl$ | Failure | $T^{fl}$ | $\tilde{fl}$ | $u^{fl}$ | $h$++, $u^{fl}$-- | |
| $frp$ | Finish of repair | $T^{rp}$ | $srp$ | $u^{frp}$ | $h$--, $u^{frp}$-- | |

**Table 5.** Events to simulate G/G/1 with failure.

The MPR proposed in this work is as follows.

Constraints (A1)-(A7), (C1)-(C4) and (B1) (B2) are applied to all events. Events $a\tilde{r}r$, $ss$, $\tilde{fl}$ and $srp$ are all zero-delay, constraints (B3) to (B11) should be applied, and constraints (B12) should be applied to events $arr$, $sf$, $fl$ and $frp$. Those constraints have been explained in the previous two examples, and then this example will not expand them.

$$\begin{aligned} (A1)-(A7),(B1),(B2),(C1)-(C4) \quad & \forall \xi \in \{a\tilde{r}r, ss, \tilde{fl}, srp\} \\ & \forall \xi \in \{arr, sf, fl, frp\} \\ (B3)-(B11) \quad & \forall \xi \in \{a\tilde{r}r, ss, \tilde{fl}, srp\} \\ (B12) \quad & \forall \xi \in \{arr, sf, fl, frp\} \end{aligned}$$

Constraints (40) to (53) are group-D constraints for canceling event $sf$. Binary variable $z_k^{\bar{sf}}$ is equal to one if executions of $sf$ are removed from the future event list, otherwise the execution is not canceled. Specifically, constraints (40) and (42) show that if $sf$ is canceled, state variable $h$ must be greater than or equal to one, and there is at least one execution of $sf$ in the future event list. Constraints (41) indicate that if $v^{\bar{sf},h,0}$ is equal to one, the machine is in up state. Constraints (43) indicate that if $v^{\bar{sf},g,0}$ is equal to one, there is no executions of $sf$ in the future event list. Constraints (44) show that if executions of $sf$ are not canceled, i.e., $z_k^{\bar{sf}}$ is equal to zero, either the

machine in in up state or there is no execution in the event list.

$$u_k^h \geq z_k^{\bar{sf}} \qquad \forall\ k \in \mathbb{K} \tag{40}$$

$$-u_k^h \geq v_k^{\bar{sf},h,0} - 1 \qquad \forall\ k \in \mathbb{K} \tag{41}$$

$$u_{k-1}^g \geq z_k^{\bar{sf}} \qquad \forall\ k \in \mathbb{K} \tag{42}$$

$$-u_{k-1}^g \geq v_k^{\bar{sf},g,0} - 1 \qquad \forall\ k \in \mathbb{K} \tag{43}$$

$$1 - z_k^{\bar{sf}} \leq v_k^{\bar{sf},s,0} + v_k^{\bar{sf},g,0} \qquad \forall\ k \in \mathbb{K} \tag{44}$$

Integer variables $k_i^{sf,0}$ and $k_i^{sf,1}$ denote the iteration index when the $i$-th execution of $sf$ is scheduled and executed, as in constraints (45) and (46).

$$k_i^{f,1} = \sum_{k \in \mathbb{K}} k w_{k,i}^{sf} \qquad \forall\ i \in \mathbb{I}^{\xi} \tag{45}$$

$$k_i^{f,0} = \sum_{k \in \mathbb{K}} k x_{k,i}^{sf} \qquad \forall\ i \in \mathbb{I}^{\xi} \tag{46}$$

Binary variable $\theta_{i,k}^{sf}$ equal to one represents that $i$-th execution of event $sf$ is canceled in iteration $k$. Constraints (47) and (48) state that if the $i$-th execution of $sf$ is canceled in iteration $k$, then it must be scheduled before iteration $k$ and executed after iteration $k$. Binary variable $\phi_{i,k}^{sf,0}$ equal to one indicates that the condition to cancel $sf$ is not true in iteration $k$ or the $i$-th execution of $sf$ is scheduled after iteration $k$, as in constraints (49). Binary variable $\phi_{i,k}^{sf,1}$ equal to one indicates that the $i$-th execution of $sf$ is executed before iteration $k$, as in constraints (50). Constraints (51) state that if execution $i$ of $sf$ is not canceled in iteration $k$, it is either because the condition to cancel is false, or it is executed before or scheduled after iteration $k$.

$$k z_k^{\bar{sf}} - k_i^{sf,0} - 1 \geq K(\theta_{i,k}^{sf} - 1) \qquad \forall\ i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \tag{47}$$

$$k_i^{sf,1} - 1 - k z_k^{\bar{sf}} \geq k(\theta_{i,k}^{sf} - 1) \qquad \forall\ i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \tag{48}$$

$$k_i^{sf,0} - k z_k^{\bar{sf}} \geq k(\phi_{i,k}^{sf,0} - 1) \qquad \forall\ i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \tag{49}$$

$$k z_k^{\bar{sf}} - k_i^{sf,1} \geq K(\phi_{i,k}^{sf,1} - 1) \qquad \forall\ i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \tag{50}$$

$$1 - \theta_{i,k}^{sf} \leq \phi_{i,k}^{sf,0} + \phi_{i,k}^{sf,1} \qquad \forall\ i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \tag{51}$$

Binary variable $\gamma_{i,k}^{sf}$ equal to one represents that the $i$-th execution of event $sf$ is performed in iteration $k$, and it is not canceled, as in constraints (53). Constraints (52) state that if $\gamma_{i,k}^{sf}$ is equal to zero, it is either because the execution is not performed in iteration $k$ or it is canceled.

$$\gamma_{i,k}^{sf} \geq w_{i,k}^{sf} - \sum_{k' \in \mathbb{K}} \theta_{i,k'}^{sf} \qquad \forall\ i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \tag{52}$$

$$w_{i,k}^{sf} - \sum_{k' \in \mathbb{K}} \theta_{i,k'}^{sf} - 1 \geq (N^{fl} + 1)(\gamma_{i,k}^{sf} - 1) \qquad \forall\ i \in \mathbb{I}^{\xi}, k \in \mathbb{K} \tag{53}$$

Constraints (54) to (60) depict the evolution of state variables. Constraints (54) to (57), referring to constraints (E5) and (E6), are similar to the two examples previously discussed, state variables $u^{arr}$, $q$, $u^{fl}$ and $u^{frp}$ are all changed by events without cancellation. Constraints (58) state that if event $sf$ is canceled in iteration $k$, the number of executions of $sf$ in the future event list will be reduced to zero. Constraints (59) and (60) show that if $sf$ is not canceled in iteration $k$, the state variable $g$ will be increased by one if $ss$ is executed, and decreased by one if $sf$ is executed without being canceled. The rest of the model indicates the initialization and range of state variables.

$$u_k^{arr} = u_{k-1}^{arr} + \sum_{i \in \mathbb{I}^j} w_{i,k}^{\tilde{arr}} - \sum_{i \in \mathbb{I}^j} w_{i,k}^{arr} \qquad \forall\ k \in \mathbb{K} \tag{54}$$

$$u_k^{q} = u_{k-1}^{q} + \sum_{i \in \mathbb{I}^j} w_{i,k}^{arr} - \sum_{i \in \mathbb{I}^j} w_{i,k}^{ss} \qquad \forall\ k \in \mathbb{K} \tag{55}$$

$$u_k^{fl} = u_{k-1}^{fl} + \sum_{i \in \mathbb{I}^j} w_{i,k}^{\tilde{fl}} - \sum_{i \in \mathbb{I}^j} w_{i,k}^{fl} \qquad \forall\ k \in \mathbb{K} \tag{56}$$

$$u_k^{frp} = u_{k-1}^{frp} + \sum_{i \in \mathbb{I}^j} w_{i,k}^{srp} - \sum_{i \in \mathbb{I}^j} w_{i,k}^{frp} \qquad \forall\ k \in \mathbb{K} \tag{57}$$

$$u_k^{g} \leq 1 - z_k^{\bar{sf}} \qquad \forall\ k \in \mathbb{K} \tag{58}$$

$$u_k^{g} \leq u_{k-1}^{g} - \sum_{i \in \mathbb{I}} \gamma_{i,k}^{sf} + \sum_{i \in \mathbb{I}} w_{i,k}^{ss} + z_k^{\bar{sf}} \qquad \forall\ k \in \mathbb{K} \tag{59}$$

$$u_k^{g} \geq u_{k-1}^{g} - \sum_{i \in \mathbb{I}} \gamma_{i,k}^{sf} + \sum_{i \in \mathbb{I}} w_{i,k}^{ss} - z_k^{\bar{sf}} \qquad \forall\ k \in \mathbb{K} \tag{60}$$

$$u_0^{arr} = u_0^{fl} = u_0^{r} = u_0^{g} = u_0^{q} = 0$$

$$u_k^{arr}, u_k^{fl}, u_k^{frp}, u_k^{g} \in \{0,1\},\ u_k^{q} \in \{0, ..., Q\}$$

Given the set of state variables, the set of zero-delay events, the sets of positive-delay events, as in Tables 2, 4, 5, and also $K$ and $N^{\xi}$, the total number of iterations and the maximum number of executions of each event, respectively, the implementation of the proposed approach is trivial. Appendix I provides the pseudo code of the proposed approach. As it can be seen, the algorithm is composed of several for-loops only. The java source code of an automatic MPR generation procedure can be found on the repository at https://github.com/myrazhang/MPRofDES.

## 8. Discussion

The MPR proposed in this work differs from the state-of-the-art MPR (Chan and Schruben (2008)) in the objective function. In the state-of-the-art formulation, the constraints represent the event triggering relationships, i.e., the execution time of the triggered event must be later than that of the triggering event with an occurrence delay. Hence, the objective function must be the minimization of the execution time of all the events, so that all the events are executed as soon as possible. In this work, the constraints represent that once the condition to schedule or cancel an event is true, the event must be scheduled or canceled, i.e., the logic that events must be executed as soon as possible is forced by constraints. Thus, the objective function can be defined

in a more flexible way.

The proposed MPR presents only the realization of a sample path of DES, since it takes the samples of random variates as the coefficients of constraints. However, expanding the MPR from single-sample-path model into a multiple-sample-path model can be achieved by adding one more subscript to each decision variables, for instance, $\mathcal{E}_k$ replaced by $\mathcal{E}_{k,r}$ where $r$ represents $r$-th replicates. The resulting MPR is separable since there are not constraints linking variables from different sample paths, i.e., the sample paths are independent from each other. Multiple-sample-path models can be plugged into retrospective algorithms so that optimization problems considering sampling noise can be solved.

As it can be seen, the proposed MPR contains both integer variables and real-valued variables. Furthermore, if the steady state performance from simulation model is of interest, the simulation length, i.e., the number of event executions, is usually long. Thus, the MPR has many variables and constraints. For instance, the MPR of G/G/m model with simulation length equal to 1000 jobs contains 12,000 real-valued and more than 32,000,000 integer variables and more than 64,000,000 constraints. It is more reasonable to use the MPR as a white-box representation of DES, other than solving it with an optimizer because the computational complexity is extremely high.

An application of the MPR is the calculation of gradient/sub-gradient from a simulation run, based on the sensitivity analysis of an approximate linear programming (LP) of the MPR. The procedure is as follows. The DES is first simulated with an event-scheduling execution logic, and the optimal solution of the MPR can be obtained during the simulation run. The approximate LP is formulated by replacing all the decision variables with a big-M multiplier, i.e., all variables except the ones representing event scheduling times, execution times, and state variables, with the value in the optimal solution, and relaxing the integrality of the remaining variables. By solving the dual problem of the approximate LP, an approximate gradient/sub-gradient of the objective function can be obtained on some coefficients (for instance, parameters of the distribution of occurrence delay $T^{\xi}$, or the thresholds $a^{\xi,s}$ and $c^{\xi,s}$ of the ranges that compose the conditions to schedule or cancel events, which are part of the conditions to schedule events). Under the black-box settings, deriving the gradient/sub-gradient must exploiting the neighborhood with several simulation runs, and the number of simulation runs needed to get the gradient of one point grows exponentially on the dimension, thus the proposed white-box MPR of DES can save the computational time for simulation.

Another way to make use of the MPR is to derive an MPR of a simulation optimization problem, as previously introduce while discussing objective function flexibility. In the applications in manufacturing and service operations, the decision variables of an optimization problem are usually the thresholds $a^{\xi,s}$ and $c^{\xi,s}$ of the ranges that compose the conditions to schedule or cancel events, which are part of the conditions to schedule events. So, to transform the MPR of simulation into MPR of simulation optimization, one just needs to specify that certain thresholds are decision variables instead of coefficients. Then, the system performance could be in the objective function or in a constraint stating that it must achieve a target. The convenience of transforming the proposed MPR into an MPR of simulation optimization represents another contribution of this work, and simulation-based optimization techniques enriched with the MPR can be developed in the future.

## 9. Conclusion

This work proposes an MPR of a sample path of a DES model that satisfies certain assumptions, based on the widely applied event-scheduling execution logic of DES. In the MPR, the decision variables include event scheduling times, execution times, and system state after each event execution, and the constraints represent that events are scheduled or canceled when the system state satisfies the condition to schedule or cancel. Furthermore, the MPR also takes the samples of random variates as the coefficients of constraints, where the random variates are usually the time delay between the occurring time and scheduling time of an event execution. The equivalence of the MPR and the simulation run implemented with an event-scheduling algorithm has been validated using several cases.

The proposed approach enriches the DES with an equivalent white-box MPR. In the literature, most of the simulation-based optimization techniques consider DES as black-box, which usually requires intensive simulation budget. Based on this work, it will be possible to develop optimization techniques taking advantage of the white-box representation. Future work will be dedicated to developing such research direction, such as efficient gradient/sub-gradient generation algorithms or optimization approaches to solve design or operation problems in manufacturing and service systems, for instance finding the optimal capacities of the queues or the control policy.

## References


Alfieri, A., & Matta, A. (2012). Mathematical programming formulations for approximate simulation of multistage production systems. *European Journal of Operational Research*, *219*(3), 773–783.

Alfieri, A., Matta, A., & Pastore, E. (2020). The time buffer approximated buffer allocation problem: A row–column generation approach. *Computers & Operations Research*, *115*, 104835.

Basile, F., Chiacchio, P., & De Tommasi, G. (2012). On k-diagnosability of petri nets via integer linear programming. *Automatica*, *48*(9), 2047–2058.

Bemporad, A., & Morari, M. (1999). Control of systems integrating logic, dynamics, and constraints. *Automatica*, *35*(3), 407–427.

Chan, W. K., & Schruben, L. (2008). Optimization models of discrete-event system dynamics. *Operations Research*, *56*(5), 1218–1237.

Di Marino, E., Su, R., & Basile, F. (2020). Makespan optimization using timed petri nets and mixed integer linear programming problem. *IFAC-PapersOnLine*, *53*(4), 129–135.

Law, A. M. (2014). *Simulation modeling and analysis* (Vol. 5). McGraw-Hill New York.

Matta, A. (2008). Simulation optimization with mathematical programming representation of discrete event systems. In *2008 winter simulation conference* (pp. 1393–1400).

Pedrielli, G., Alfieri, A., & Matta, A. (2015). Integrated simulation–optimisation of pull control systems. *International Journal of Production Research*, *53*(14), 4317–4336.

Tan, B. (2015). Mathematical programming representations of the dynamics of continuous-flow production systems. *IIE Transactions*, *47*(2), 173–189.

Weiss, S., & Stolletz, R. (2015). Buffer allocation in stochastic flow lines via sample-based optimization with initial bounds. *OR spectrum*, *37*(4), 869–902.

Zeigler, B. P., Muzy, A., & Kofman, E. (2018). *Theory of modeling and simulation: discrete event & iterative system computational foundations*. Academic press.

Zhang, M., & Matta, A. (2020). Models and algorithms for throughput improvement problem of serial production lines via downtime reduction. *IISE Transactions*, 1–15.

Zhang, M., Matta, A., & Alfieri, A. (2020). Sample-path algorithm for global optimal solution

of resource allocation in queueing systems with performance constraints. In *2020 winter simulation conference (wsc)* (pp. 2767–2778).

Zhang, M., Pastore, E., Matta, A., & Alfieri, A. (2021). Buffer allocation problem in production flow lines: a new benders decomposition based exact solution approach. *under review*.

## Appendix I: implementation pseudo code

The pseudo code below shows the implementation of the proposed approach. The algorithm requires $\mathbb{X}$, $\mathbf{s}_0$, $\mathbb{S}^{\xi}$, $\mathbb{S}^{\bar{\xi}}$, $\mathbb{S}^{c}$, $K$ and $N^{\xi}$ as the input. In the main implementation of the algorithm, each constraint introduced in Section 2.4 is written inside certain for-loop. The expression of the constraints depends on the grammar of the state-of-the-art optimizer one want to use, such as Cplex, Gurobi, etc.

**Algorithm 2** Implementation pseudo code.

**Input:**

$\mathbb{X}$: set of events.
$\mathbf{s}_0$: initial system state.
$\mathbb{S}^{\xi}$: set of state variables included in the condition to schedule event $\xi$.
$\mathbb{S}^{\bar{\xi}}$: set of state variables included in the condition to cancel event $\xi$.
$\mathbb{S}^{c}$: set of counting variable.
$K$: number of iterations until the termination of simulation.
$N^{\xi}$: execution number of event $\xi$.

(A6) (E7)
**for** $\xi \in \mathbb{X}$ **do**
  **for** $i = 1$ *to* $N^{\xi}$ **do**
    **for** $k = 1$ *to* $K$ **do**
      (A1) (A2) (B1) (B2)
    **end for**
    (A3) (A7) (C2) (C3)
  **end for**
  **for** $i = 1$ *to* $N^{\xi} - 1$ **do**
    (C1) (C4)
  **end for**
**end for**
**for** $k = 1$ *to* $K$ **do**
  (A4) (A5)
**end for**
**for** each zero-delay event $\xi \in \mathbb{X}$ **do**
  **for** $k = 0$ *to* $K - 1$ **do**
    **for** $s \in \mathbb{S}^{\xi}$ **do**
      (B3) (B4) (B5) (B6)
    **end for**
    (B7) (B9) (B10)
  **end for**
  (B8)
  **for** $i = 1$ *to* $N^{\xi}$ **do**
    (B11)
  **end for**
**end for**

**for** each positive-delay event $\xi \in \mathbb{X}$ **do**
  **for** $k = 0$ *to* $K - 1$ **do**
    **for** $i = 1$ *to* $N^{\xi}$ **do**
      (B12) (D6) (D7) (D8) (D9) (D10) (D11) (D12) (D13) (D14)
    **end for**
    **for** $s \in \mathbb{S}^{\bar{\xi}}$ **do**
      (D1) (D2) (D3) (D4)
    **end for**
    (D5)
  **end for**
**end for**
**for** $k = 0$ *to* $K - 1$ **do**
  **for** $s \in \mathbb{S}/\mathbb{S}^c$ **do**
    (E1)
  **end for**
  **for** $s \in \mathbb{S}^c$ **do**
    (E2) (E3) (E4)
  **end for**
**end for**

## Appendix II: Event-scheduling algorithm

**Algorithm 3** Event-scheduling algorithm.

**Input:**
Set of events $\mathbb{X}$.
Initial system state: $\mathbf{s}_0$.
Simulation length: number of iterations $K$.
Simulation clock: $clock = 0$.
Number of scheduled executions of event type $\xi$: $i^{\xi} = 0$.
*FutureEventList* $= \{\}$.

**for** k=0 to K-1 **do**
  **Schedule new executions**
  **for** each event $\xi \in \mathbb{X}$ **do**
    **if** $CS^{\xi}(\mathbf{s}_k)$ **then**
      $i^{\xi} = i^{\xi} + 1$
      Sample occurring delay $t^{\xi}_{i^{\xi}}$ from random variate $T^{\xi}$
      Add execution $(\xi, i^{\xi}, clock, clock + t^{\xi}_{i^{\xi}})$ to *FutureEventList*.
    **end if**
  **end for**

  **Cancel executions**
  **for** each execution *exe*$\in$*FutureEventList* **do**
    **if** $CC^{exe.event\_type}(\mathbf{s}_k)$ **then**
      Remove *exe* from *FutureEventList*
    **end if**
  **end for**

  **Perform the earliest execution**

From *FutureEventList*, take execution $exe$, which is with the earliest occurring time.
Update system state: $\mathbf{s}_{k+1} \leftarrow F^{exe.event\_type}(\mathbf{s}_k)$.
$clock = exe.occurring\_time$.
**end for**

**Appendix III: Modified event-scheduling logic.**

**Algorithm 4** Modified event-scheduling logic.

**for** k=0 to K-1 **do**
  **Schedule new zero-delay executions**
  **for** each zero-delay event $\xi \in \mathbb{X}$ **do**
    **if** $CS^{\xi}(\mathbf{s}_k)$ & there is no execution of event $\xi$ in *FutureEventList* **then**
      $i^{\xi} = i^{\xi} + 1$
      Add execution $(\xi, i^{\xi}$,clock,clock, $false)$ to *FutureEventList*.
    **end if**
  **end for**

  **Cancel executions**
  **for** each positive-delay execution $exe \in$ *FutureEventList* **do**
    **if** $CC^{\xi}(\mathbf{s}_k)$ **then**
      $exe.cancel$ = true, $u^{\xi} = 0$.
    **end if**
  **end for**

  **Perform an execution**
  From *FutureEventList*, take execution $exe$, which is with the earliest occurring time.
  $clock = exe.occurring\_time$.
  **if** $T^{\xi} > 0$ **then**
    **if** $exe.cancel$ is false **then**
      Update system state: $\mathbf{s}_{k+1} \leftarrow F^{\xi}(\mathbf{s}_k)$.
    **end if**
  **else**
    Update system state: $\mathbf{s}_{k+1} \leftarrow F^{\xi}(\mathbf{s}_k)$.
    **Schedule new positive-delay execution**
    **if** $\xi$ is the counting event of $\xi^{'}$ **then**
      Sample $t_i^{\xi^{'}}$ from random variate $T^{\xi^{'}}$.
      Add execution $(\xi^{'}, i$,clock,clock+$t_i^{\xi^{'}}, false)$ to *FutureEventList*.
    **end if**
  **end if**
**end for**