A MANUFACTURING SYSTEM AS A HYBRID SYSTEM

Shirantha Welikala, Christos G. Cassandras

June-2019

Boston University

Division of Systems Engineering

Technical Report

BOSTON UNIVERSITY

A MANUFACTURING SYSTEM AS A HYBRID SYSTEM

Shirantha Welikala, Christos G. Cassandras

Boston University
Division of Systems Engineering
15 Saint Mary's Street
Boston, MA 02215
www.bu.edu/se

June-2019

Technical Report

This report serves as a reference manual to the hybrid system simulator available at http://scc-lite.bu.edu/ shiran27/HSSimV2/. This contains a brief problem formulation section and a summary of the solution technique used along with some key references. Also, this report explains each and every component of the simulator and provides few illustrative examples on the usage of the simulator.

Contents

1 Problem Formulation and Solution

In this work, a single server queuing system model with service times determined by time driven dynamics is used to model a typical single stage manufacturing system. Due to the simultaneous operation of both event-driven dynamics and the time-driven dynamics, this manufacturing system model as shown in 1 is essentially a hybrid system.

In section 1.1, we will present the considered problem in a more general setting. The consequent section 1.2 discuss the solution technique in a general sense. Then in the section 1.3, the specific problem that has been implemented in the simulator is presented. Section 1.4 then presents the details of the solution scheme implemented in the simulator.

1.1 Generalized problem formulation

A sequence of N jobs denoted by $C_1, C_2, \cdots, C_N$ are assigned by an external source to arrive for processing at known (arrival) times $a_1, a_2, \cdots, a_N$. The jobs are processed in a first-come-first-serve (FCFS) basis by a work conserving and non-preemptive server. The service time of the job C_i is $s_i(u_i)$, which is a function of the control input u_i and the specific job C_i itself. In this analysis, the control input applied to a job is constrained to be a constant over the duration of its service period. However, the server is allowed to use different control inputs for different jobs.

Time driven dynamics. Each job (say C_i) at some common initial time $t = x_0$ is considered to be at a certain physical state ζ. This can be expressed as $z_i(x_0) = zeta_i$ where z_i denotes the time varying physical state of the job. Subsequently, this physical state $z_i(t)$ of the job C_i evolves according to the time driven dynamics given by,

$$\dot{z}_i(t) = g_i(z_i, u_i, t), \text{ with } z_i(x_0) = \zeta_i. \tag{1}$$

Event driven dynamics. The completion time (or the departure time) of the each job is denoted by x_i and it is given by the standard Lindley's equation for a FCFS

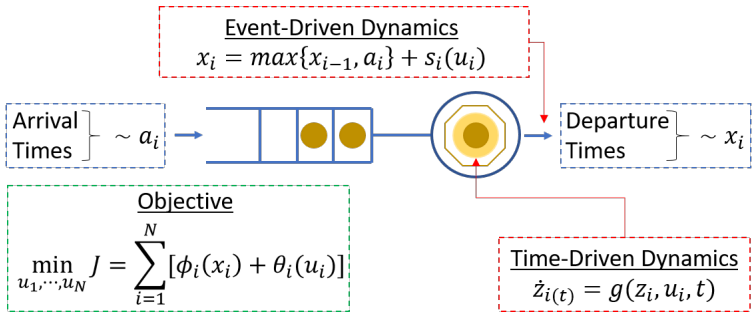

Figure 1: Single Server Queuing System

non-preemptive queue [1]:

$$x_i = \max\{x_{i-1}, a_i\} + s_i(u_i), \text{ with } x_0 = -\infty \tag{2}$$

Note that the control input choice u_i affects both the physical state z_i and the next temporal state x_i. In this framework, a job C_i is considered to require a service time $s_i(u_i)$ such that its physical state z_i reaches a certain quality level denoted by q_i. Therefore, the processing time required for a job can be formally defined as,

$$s_i(u_i) = \min\{t \geq 0 : z_i(x_0 + t) = \int_{x_0}^{x_0+t} g_i(z_i, u_i, \tau)d\tau + z_i(x_0) \in q_i\}. \tag{3}$$

Objective function. For the considered hybrid system defined by the equations (1), (2) and (3), our aim is to find the control sequence $\{u_1, u_2, \cdots, c_N\}$ which minimizes the objective function of the form

$$J = \sum_{i=1}^{N} [\phi_i(x_i) + u_i(u_i)]. \tag{4}$$

Regarding this objective function format, one important thing to notice is that it does not explicitly involve the continuously varying physical state variables $z_i(t)$. However, note that the terminal physical state of each job plays a role in defining the service time (in (3), $s_i(u_i)$ is such that $z_i(x_i) \in q_i$), and hence it indirectly directly affects the departure time x_i. Therefore, by appropriately choosing the functions $\phi_i(\cdot)$ and $s_i(\cdot)$ we can indirectly impose a cost on the physical state.

Optimal control problem. For this problem framework, the complete optimal control problem, denoted by **P** can be stated in the following form:

$$\begin{aligned} \mathbf{P}: \min_{u_1,u_2,\cdots,u_N} & \left\{ J = \sum_{i=1}^{N} \left\{ \phi_i(x_i) + \theta_i(u_i) \right\} \right\} \\ \text{Subject to} & \\ x_i &= \max\{x_{i-1}, a_i\} + s_i(u_i), && \text{for } i = 1, 2, \cdots, N \text{ and,} \\ s_i(u_i) &\geq 0 && \text{for } i = 1, 2, \cdots, N. \end{aligned} \tag{5}$$

1.2 Generalized solution technique

Sub-optimal problem. In the works [2] and [3], an algorithm named **Forward Algorithm** (which will be presented later) is proposed to solve the above optimal control problem **P**. This is achieved by iteratively solving a separate sub-problem denoted by $Q(k, n)$ (also called the sub-optimal problem) which intends to find the

control inputs u_i (or the service times $s_i(u_i)$) for the jobs $C_k, C_{k+1}, \cdots, C_{n-1}, C_n$. This sub-optimal problem is given by,

$$\begin{aligned}\mathbf{Q}(\mathbf{k},\mathbf{n}): \min_{u_k,\cdots,u_n} &\left\{ J(k,n) = \sum_{i=k}^{n} \left\{ \phi_i(x_i) + \theta_i(u_i) \right\} \right\} \\ &\text{Subject to} \\ x_i =& a_k + \sum_{j=k}^{i} s_j(u_j), \text{ for } i = k, k+1, \cdots, n, \\ x_i \geq& a_{i+1}, \text{ for } i = k, k+1, \cdots, n-1 \text{ and}, \\ s_i(u_i) \geq& 0, \text{ for } i = k, k+1, \cdots, n.\end{aligned} \tag{6}$$

The constraints of this sub optimal problem implies that jobs $C_k, C_{k+1}, \cdots, C_{n-1}$ should arrive during the server is busy. However the job C_n is not constrained in a such manner, therefore, it can either end the busy period (i.e. $x_n < a_{n+1}$) or continue it (i.e. if $x_n \geq a_{n+1}$). Since this x_n represents a part of the optimal solution of $Q(k,n)$, it is more formally denoted as $x_n^*(k,n)$.

Also, note that all the x_i terms can be replaced by control input u_i terms using the first equality constraint. Therefore, the problem $Q(n,k)$ can be written in a compact form (such that it only involves the $s_i(u_i)$ terms) as,

$$\begin{aligned}\mathbf{Q}(\mathbf{k},\mathbf{n}): \min_{u_k,\cdots,u_n} &\left\{ J(k,n) = \sum_{i=k}^{n} \left\{ \phi_i(a_k + \sum_{j=k}^{i} s_j(u_j)) + \theta_i(u_i) \right\} \right\} \\ &\text{Subject to} \\ a_k + \sum_{j=k}^{i} s_j(u_j) \geq& a_{i+1}, \text{ for } i = k, k+1, \cdots, n-1 \text{ and}, \\ s_i(u_i) \geq& 0 \text{ for } i = k, k+1, \cdots, n.\end{aligned} \tag{7}$$

The forward algorithm. In order to solve the problem **P** stated in (5), the simulator uses the "**Forward Algorithm 1**" proposed in the work [2]. It is shown in Algorithm 1. Also, it is important to point out that we need to make a few basic assumptions for this method to work. Those assumptions are typically related to the shape of the objective function components $\phi_i(\cdot)$ and $\theta_i(\cdot)$. More details can be found in [2].

In there, the step 5 shows how the optimal control inputs $u_1^*, u_2^*, \cdots, u_N^*$ are found via solving multiple $Q(k,n)$ sub-optimal problems. Note that the solution of sub-optimal problem $Q(k,n)$ is typically denoted as $u_k^*(k,n), u_{k+1}^*(k,n), \cdots, u_n^*(k,n)$.

It should be noted that there are execution time wise improved versions of this algorithm such as the "**Forward Algorithm 2**" found in the same work [2] and the two algorithms proposed in [3]. However, all of these algorithms essentially use the

Algorithm 1 Forward Algorithm 1

Initialize with $k = 1, n = 1, a_{N+1} = \infty, x_0 = -\infty$;
while $n \leq N$ **do**
  Solve sub-optimal problem $Q(k, n)$;
  if $x_n^*(k, n) < a_{n+1}$ **then**
    $u_j^* \leftarrow u_j^*(k, n)$ for $j = k, \cdots, n$
    $k \leftarrow n + 1$
  end if
  $n \leftarrow n + 1$
end while

same sub-optimal problem $Q(k, n)$ stated in (7) to get to the optimal control policy. Therefore, solving the sub-optimal problem plays a key role in implementation.

1.3 The specific problem considered in the simulator

In this section, we introduce the specific form of the problem which is addressed by the developed simulator.

Time driven dynamics. The time driven dynamics, in other words, the nature of the evolution of the physical state z_i with respect to the control input u_i is given by,

$$\dot{z}_i(t) = u_i \text{ with } z_i(x_i - s_i(u_i)) = 0 \text{ and } z_i(x_i) = q_i. \tag{8}$$

This represents a situation where the physical state z_i of the job C_i increases at a rate determined by the control input u_i. The initial and terminal conditions for z_i are 0 and q_i. And those happens respectively at the beginning of the service time (i.e. at $t = x_i - s_i(u_i)$) and at the departure time (i.e. at $t = x_i$).

Event driven dynamics. The relationship between the service time $s_i(u_i)$ of a job C_i and its control input u_i is taken to have the form,

$$s_i(u_i) = \frac{q_i}{u_i} \tag{9}$$

Here the q_i term (which we also saw in (8)) represents a pre-assigned parameter which can be thought of as the "**Required Quality Level**" of the job C_i. From (9) it is clear that the service time is proportional to the required quality level and it is inversely proportional to the control input. Therefore it is a realistic situation.

Now, the Lindley's equation can be written as,

$$x_i = \max\{x_{i-1}, a_i\} + \frac{q_i}{u_i} \text{ with, } x_0 = -\infty. \tag{10}$$

Objective function. In the simulator, the considered objective function takes the form,

$$J = \sum_{i=1}^{N} \{\alpha x_i^2 + \beta u_i^2 + \gamma w_i^2\}, \tag{11}$$

where α, β and γ stands for the penalty coefficients on departure times (x_i), control inputs (u_i) and waiting times (w_i). The waiting time was taken as $w_i = x_i - a_i$.

Optimal control problem. The complete optimal control problem, denoted by **P** can be stated in the following form:

$$\mathbf{P}: \min_{u_1, u_2, \cdots, u_N} \left\{ J = \sum_{i=1}^{N} \left\{\alpha x_i^2 + \beta u_i^2 + \gamma w_i^2\right\} \right\}$$

Subject to

$$\begin{aligned} x_i &= \max\{x_{i-1}, a_i\} + \frac{q_i}{u_i}, && \text{for } i = 1, 2, \cdots, N \text{ and,} \\ \frac{q_i}{u_i} &\geq 0 && \text{for } i = 1, 2, \cdots, N. \end{aligned} \tag{12}$$

1.4 The solving technique used in the simulator

Sub-optimal problem For the specific problem form discussed in section 1.3, as a solving technique, we use the Forward Algorithm 1 proposed in [2]. This process requires us to solve a sub-optimal problem denoted by $Q(k, n)$ given by,

$$\mathbf{Q}(\mathbf{k}, \mathbf{n}): \min_{u_k, \cdots, u_n} \left\{ J(k, n) = \sum_{i=k}^{n} \left\{\alpha x_i^2 + \beta u_i^2 + \gamma w_i^2\right\} \right\}$$

Subject to

$$\begin{aligned} x_i =& a_k + \sum_{j=k}^{i} s_j(u_j), \text{ for } i = k, k+1, \cdots, n, \\ x_i \geq& a_{i+1}, \text{ for } i = k, k+1, \cdots, n-1 \text{ and,} \\ s_i(u_i) \geq& 0, \text{ for } i = k, k+1, \cdots, n. \end{aligned} \tag{13}$$

Since using $s_j(u_j) = \frac{q_j}{u_j}$ can make the constraints non-linear, lets use $s_j = s_j(u_j)$ as the program variable. Therefore note that $u_i = \frac{q_i}{s_i}$. Now, the objective function of

$Q(k,n)$ can simplified as,

$$\begin{aligned}
J(k,n) &= \sum_{i=k}^{n} \Big\{ \alpha x_i^2 + \beta u_i^2 + \gamma w_i^2 \Big\} \Big\} \\
&= \sum_{i=k}^{n} \Big\{ \alpha x_i^2 + \beta \frac{q_i^2}{s_i^2} + \gamma (x_i - a_i)^2 \Big\} \Big\} \\
&= \sum_{i=k}^{n} \Big\{ \alpha (a_k + \sum_{j=k}^{i} s_j)^2 + \beta \frac{q_i^2}{s_i^2} + \gamma (a_k + \sum_{j=k}^{i} s_j - a_i)^2 \Big\} \Big\} \\
&= \sum_{i=k}^{n} \Big\{ \beta \frac{q_i^2}{s_i^2} + 2(\alpha a_k - \gamma(a_i - a_k)) \sum_{j=k}^{i} s_j + (\alpha + \gamma) \Big(\sum_{j=k}^{i} s_j \Big)^2 \\
&\quad + \gamma (a_i - a_k)^2 + \alpha a_k^2 \Big\} \Big\}. \qquad (14)
\end{aligned}$$

Now, $J(k,n)$ is only a function of service times s_i for $i = k, \cdots, n$. Note that $\alpha, \beta, \gamma, \{a_i\}_{i=k}^n$ and $\{q_i\}_{i=k}^n$ are all known values.

Now the sub-optimal problem can be written in a compact form as follows.

$$\begin{aligned}
&\mathbf{Q(k,n)}: \min_{u_k, \cdots, u_n} J(k,n) \\
&\text{Subject to} \\
&a_k + \sum_{j=k}^{i} s_j \geq a_{i+1}, \text{ for } i = k, k+1, \cdots, n-1 \text{ and}, \\
&\qquad s_i \geq 0, \text{ for } i = k, k+1, \cdots, n.
\end{aligned} \qquad (15)$$

Projected gradient descent method. Notice that the objective function associated with the sub-optimal problem in (15) is convex (see (14)) and the constraints are linear. Therefore, using the projected gradient descent method is the most straightforward way to reach the optimal solution: $\{s_i^*\}_{i=k}^n$.

The gradient of the objective function $J(k,n)$ given in (12) in the direction of s_m (where $m \in \{k, k+1, \cdots, n\}$) takes the form,

$$\frac{\partial J(k,n)}{\partial s_m} = -2\beta \frac{q_m^2}{s_m^3} + 2(\alpha + \gamma) \sum_{i=m}^{n} \sum_{j=k}^{i} s_j + 2(\alpha + \gamma)(n - m + 1) a_k - 2\gamma \sum_{i=m}^{n} a_i \qquad (16)$$

The projected gradient descent steps take the form,

$$s_m^{(l+1)} = \prod_{s_m \in B_m} \Big[s_m^{(l)} - \beta \frac{\partial J(k,n)}{\partial s_m} \Big] \text{ for } m \in \{k, k+1, \cdots, n\} \text{ and } l = 0, 1, 2, \cdots. \qquad (17)$$

Here, β denotes the step size, super script l denotes the update iteration number and B_m represents the constrained space (for the projection) of the variable s_m.

The used step size coefficient was $\beta = 0.005$ and typically the projected gradient descent method takes $l \simeq 90$ iterations to converge to a condition where $\sum_{m=k}^{n} \|\partial J(k,n)/\partial s_m\|^2 < 0.001$. The projection space can be defined more formally as,

$$B_m = \begin{cases} [a_{m+1} - a_k - \sum_{j=k}^{m-1} s_j, \infty] & \text{for } m \in \{k, k+1, \cdots, n-1\}, \\ [0, \infty] & \text{for } m = n. \end{cases} \tag{18}$$

The initial condition for the projected gradient descent method was generated using,

$$s_m = a_{m+1} - a_k - \sum_{j=k}^{m-1} s_j + \Delta \text{ for } m \in \{k, k+1, \cdots, n\}, \tag{19}$$

where Δ should satisfy $0 \leq \Delta$ so that the initial solution is feasible. In the implementation $\Delta = 0.5$ is used.

2 The Developed Simulator

2.1 Component wise details of the simulator

In this section, we will look at each and every component of the simulator and will describe when and how to use those components. Main component names are labelled in the Figure 2 and their respective main task is described in the table 1 along with more detailed diagrams of each component (See Figures 3-6).

Figure 2: Main Components of the Simulator

Component	Main Task
Title and Information Box	Contains a summary of the underlying problem formulation
Console Display	Important information regarding the current state of the simulator will be printed here.
Arrival time menu	Specify the arrival times of the N jobs (i.e. $\{a_i\}_{i=1}^{N}$) using this menu. There are multiple ways to generating arrival times. See Figure 3 for more details.
Quality Level Menu	Specify the required quality levels of the N jobs (i.e. $\{q_i\}_{i=1}^{N}$) using this menu. There are multiple ways to generating these quality levels. See Figure 4 for more details.
Penalty Parameter Menu	Specify the penalty parameters here (to penalize departure times, control inputs and waiting times).
Control Input Menu	Generate / compute or specify the control input values applied to each job (i.e. $\{u_i\}_{i=1}^{N}$) using this menu. See Figure 5 for more details.
Simulation menu	Use this menu to store different simulations and to compare them and also to see their real-time operation. See Figure 6 for more details.

Table 1: Caption

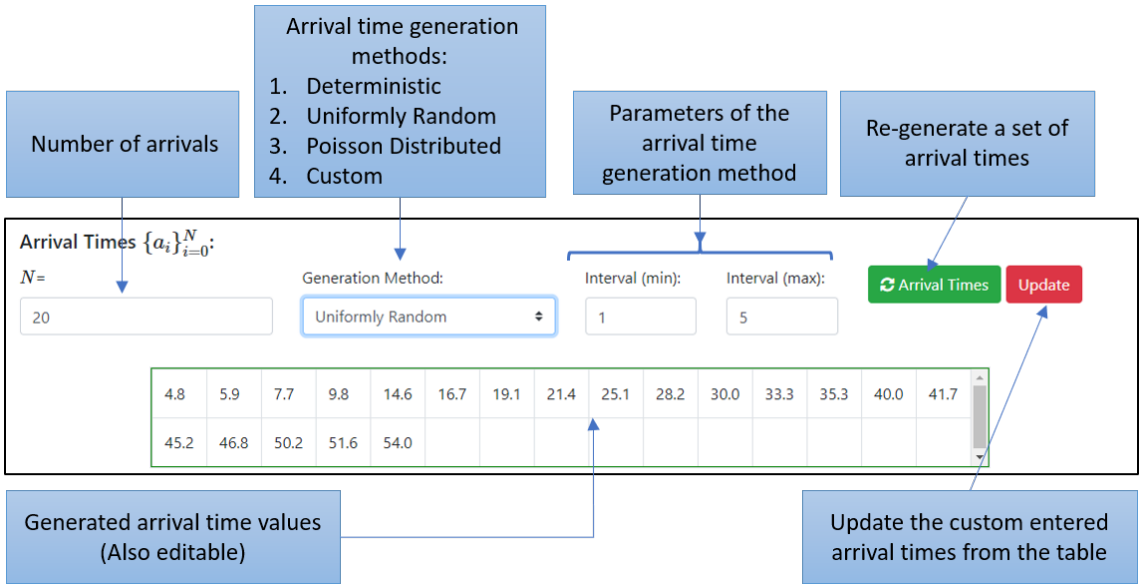

Figure 3: Arrival Time Menu

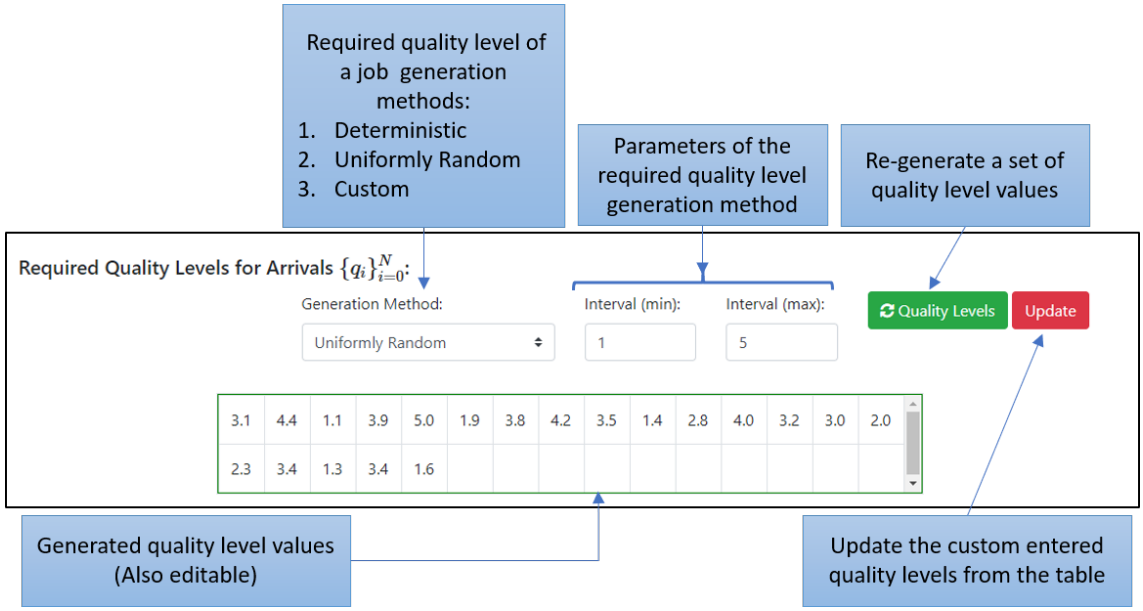

Figure 4: Quality Level Menu

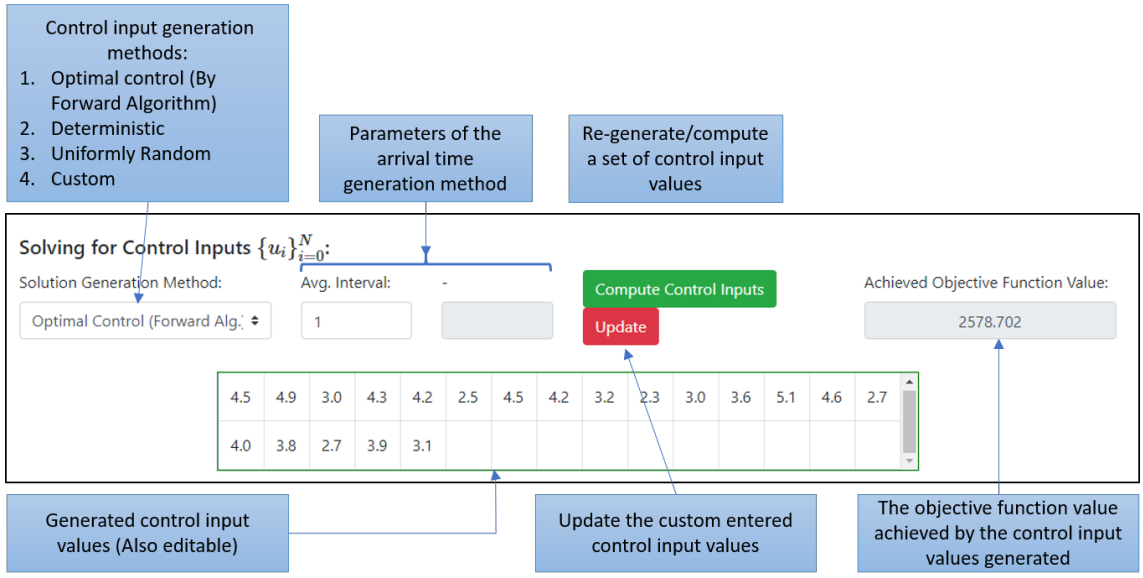

Figure 5: Control Input Menu

2.2 Some generated results

Results obtained for four case studies are illustrated in the Figures 7-10.

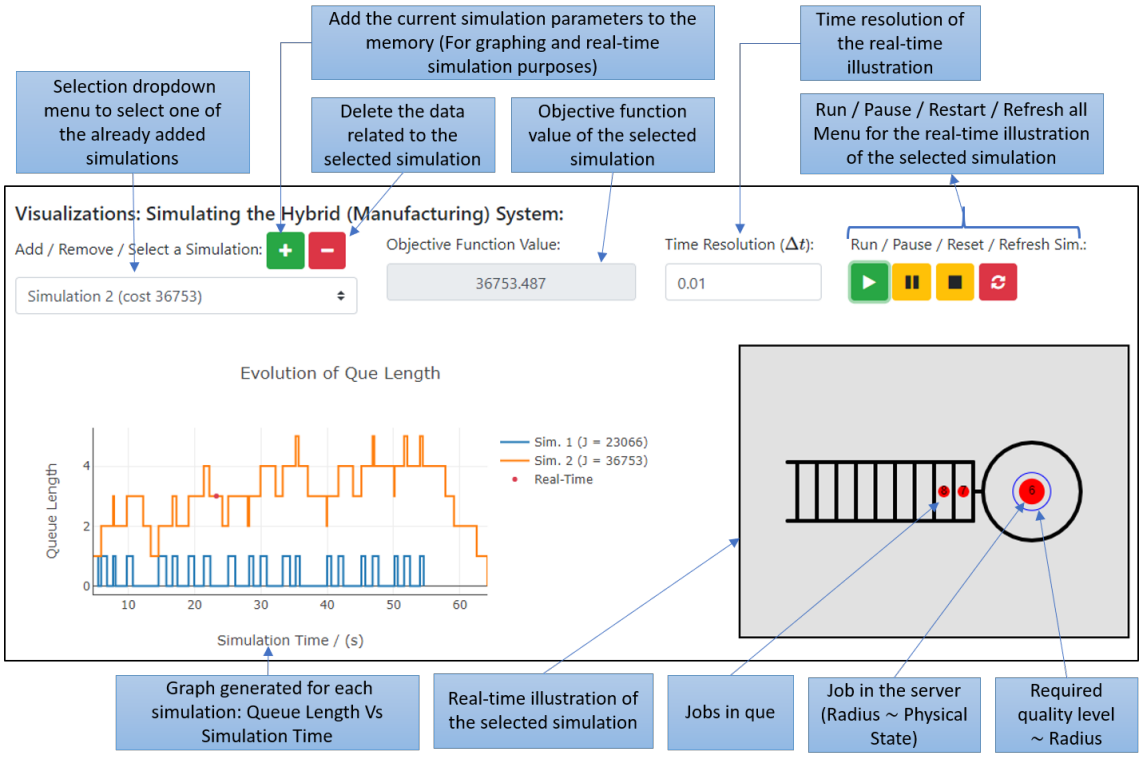

Figure 6: Simulations Menu

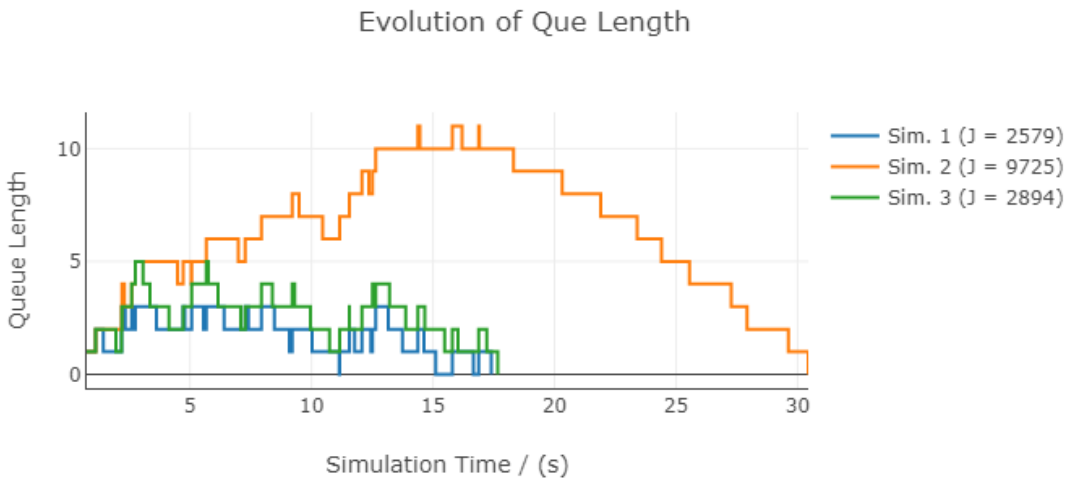

Figure 7: Case 1: Arrival times are Poisson distributed, required quality levels are uniformly distributed, all penalty parameters are equal to 1, and Sim.1 - Sim.3 respectively stands for when the control input is generated via a) Forward algorithm, b) Deterministic with $u_i = 2, \forall i$ and c) Uniformly distributed where $u_i \sim U[2, 5]$

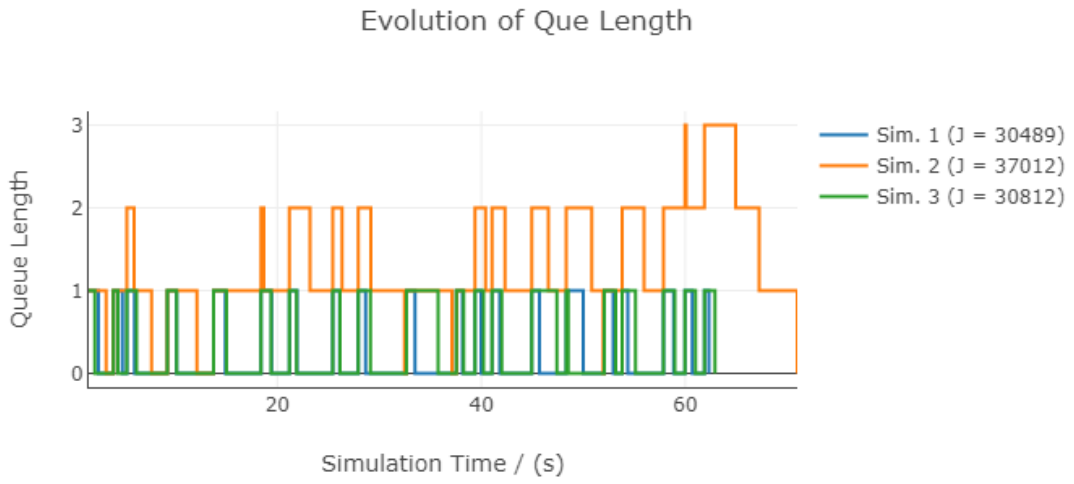

Figure 8: Case 2: Both arrival times and the required quality levels of each job is uniformly distributed, all penalty parameters are equal to 1, and Sim.1 - Sim.3 respectively stands for when the control input is generated via a) Forward algorithm, b) Deterministic with $u_i = 1, \forall i$ and c) Uniformly distributed where $u_i \sim U[1, 5]$

Figure 9: Case 2: Arrival times are deterministic, required quality levels are uniformly distributed, all penalty parameters are equal to 1, and Sim.1 - Sim.3 respectively stands for when the control input is generated via a) Forward algorithm, b) Deterministic with $u_i = 3, \forall i$ and c) Uniformly distributed where $u_i \sim U[3, 5]$

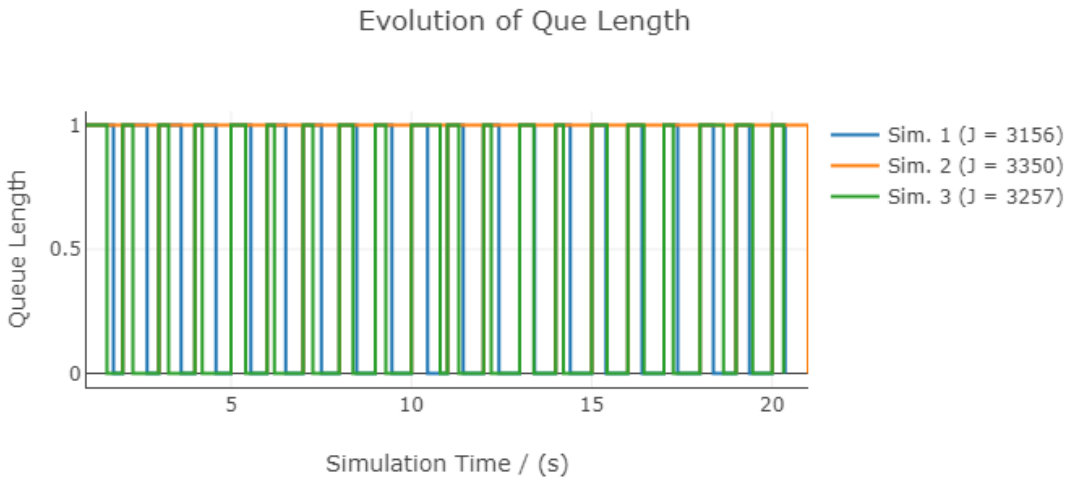

Figure 10: Case 2: Both arrival times and the required quality levels are deterministic, all penalty parameters are equal to 1, and Sim.1 - Sim.3 respectively stands for when the control input is generated via a) Forward algorithm, b) Deterministic with $u_i = 1, \forall i$ and c) Uniformly distributed where $u_i \sim U[1, 5]$

2.3 Conclusions

In general, the forward algorithm used in the simulator is capable of addressing optimal control problems defined for a single stage hybrid systems. The developed simulator uses a conventional manufacturing system as a single stage hybrid system. The control variables in this paradigm are the service times of various jobs. Moreover, the considered performance metric involves quality requirements, control inputs, delivery time requirements, and system delay time requirements. Theoretically, the considered optimal control problem is non-convex and non-differentiable due to the associated event-driven dynamics of the hybrid system. However, the used forward algorithm is a scalable and low-complexity approach to compute the optimal controls (i.e. optimal service times) for each job. Specifically, the forward algorithm is derived via identifying the busy period structure of the optimal sample path. In operation, the forward algorithm requires solving multiple, yet much smaller convex optimization problems (with linear constraints) to construct the optimal sample path. The implementation results show that the optimal controls determined by the forward algorithm are always capable of delivering better performance levels compared to when arbitrarily defined control inputs are used. In the literature, the pioneers of this forward algorithm have also proposed few improved forward algorithm versions as well [2, 3].

References

[1] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Springer Publishing Company, Inc., 2nd ed., 2010.

[2] Y. C. Cho, C. G. Cassandras, and D. L. Pepyne, "Forward algorithms for optimal control of a class of hybrid systems," in *39th IEEE Conf. on Decision and Control*, vol. 1, pp. 975–980, 12 2000.

[3] Ping Zhang and C. G. Cassandras, "An improved Forward Algorithm for optimal control of a class of hybrid systems," in *40th IEEE Conf. on Decision and Control*, vol. 2, pp. 1235–1236, 12 2001.