Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

A TIME-BASED DECOMPOSITION ALGORITHM FOR FAST SIMULATION WITH MATHEMATICAL PROGRAMMING MODELS

Arianna Alfieri

Politecnico di Torino
corso Duca degli Abruzzi 24
Torino, ITALY

Andrea Matta

Politecnico di Milano
via La Masa 1
Milano, ITALY

ABSTRACT

Mathematical programming has been proposed in the literature as an alternative technique to simulate a special class of Discrete Event Systems. Several are the benefits of using a mathematical programming model for simulating but the non–linear computational time (in the number of simulated entities) needed for the solution of the models can be a huge barrier to its use in long simulations. This paper proposes a time–based decomposition algorithm that splits the mathematical programming model into a number of submodels to be solved sequentially so as to exploit the super–additivity of many non–linear functions and make the mathematical programming approach viable also for long run simulations. The number of needed submodels is the solution of an optimization problem that minimizes the expected time to solve all the submodels. The main result is that in this way the solution time becomes a linear function of the number of simulated entities.

1 INTRODUCTION

Mathematical programming has been proposed by Schruben (2000) as an alternative technique to simulate Discrete Event Systems (DESs). Schruben represents the behavior of a $G/G/1$ system with a Linear Programming (LP) model having as objective function the sum of start and completion times of all the customers entering the system and as constraints the set of temporal inequalities modeling precedence between two consecutive customers. Starting and finishing times are the decision variables of the problem. A simplified version (with finishing times as the only decision variables) of the original Schruben's model for a $G/G/1$ system is as follows:

$$\min \quad \sum_{i=1}^{N} y_i \tag{1}$$

$$\text{s.t.} \quad y_i \geq a_i + t_i \qquad \forall i \tag{2}$$

$$y_{i+1} - y_i \geq t_{i+1} \quad i = 1,\ldots,N-1 \tag{3}$$

where N is the number of customers visiting the system. Arrival time a_i and service time t_i of each customer i (with $i = 1,\ldots,N$) are parameters of the model. They can be known from a given data set or randomly generated from some distributions. The solution of this problem provides the values for the decision variables y_i that are the completion times of each customer. Once the problem has been solved, from the values of the decision and slack variables it is possible to construct the simulated sample path and to calculate the system performance measures. On the same stream, LP models for representing tandem queuing systems and scheduling problems (Chan and Schruben 2003) (Chan and Schruben 2008) and pull control systems (Alfieri and Matta 2012b) have been proposed in the literature.

There are several benefits in using a mathematical programming model instead of a standard simulation model implemented in a computer code (Chan and Schruben 2008) (Matta 2008). Among such benefits, the possibility of easily applying sensitivity analysis and the fast convergence to a near optimal solution, when the objective function is changed to optimize some system parameters, are the most relevant ones. This last issue has been recently investigated by Alfieri and Matta (2012a).

The major drawback of mathematical programming models for simulation is the computational burden that may be encountered in some cases. The complexity of DESs forces, in most of the cases, to introduce integer variables into the mathematical programming, thus leading to unacceptable solution times (the model becomes a Mixed Integer Linear Program (MILP)). Another not affordable case is encountered with long simulations, because of the relationship between the number of decision variables and the number of simulated entities. In this case, even if the model is an LP model, the simulation time is a non–linear function (exponential in the worst case) of the number of customers simulated in the system. For instance, as N in the $G/G/1$ system increases the number of variables y_i and number of constraints in inequalities (2) and (3) also increase. Hence, the mathematical programming approach seems to be applicable only for small instances. How to deal with long runs when mathematical programming is adopted as a mean to simulate is exactly the topic of the current work.

This paper proposes an algorithm that allows extending the use of mathematical programming to long runs. The algorithm is a time–based decomposition that splits the mathematical programming model into a number of submodels that are solved sequentially one after another. The key point is exactly to create subsets of entities and solve a single model for each subset, exploiting the typical super–additivity of non–linear functions. In particular, the algorithm starts solving the first submodel, the optimal solution of which is used to initialize the second submodel and so forth until the last submodel is solved. The number of submodels is the solution of an optimization problem that minimizes the expected time to solve all the submodels. The main result is that the solution time is linearized, i.e., the time for a long simulation is a linear function of the number of simulated entities.

The proposed approach is applicable to all those systems where entities enter at the first stage and leave the system from the last stage after having visited the intermediate ones. Moreover, the scheduling of entities visiting stages is assumed to be fixed and known in advance. This class of systems is of practical interest and includes as examples transfer lines, assembly/disassembly systems, pull controlled systems.

The structure of the paper is the following. The problem we study and the developed solution method are reported in Section 2 and 3, respectively. Sections 4 and 5 contain the application of the method to specific systems. Section 6 concludes the paper.

2 PROBLEM

Given a DES the performance of which we want to study with simulation, different mathematical programming models can be used to represent its dynamic behavior (Chan and Schruben 2008). Consider an LP or MILP model that simulates the evolution of N entities in a DES. Entities can represent parts in a manufacturing system, failures in a maintenance systems, etc. The objective function of the model is the sum of the starting and finishing events occurring in the system, no matter the type of the event. These time occurrences are the decision variables of the optimization problem. Decision variables are constrained to a set of inequalities describing the system behavior. The mathematical programming model can be written in the following standard form:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{x}) = \mathbf{0} \end{aligned} \tag{4}$$

where $\mathbf{g}$ is a vector of functions describing the system constraints. Functions in $\mathbf{g}$ depend on the decision variables $\mathbf{x}$ and on the duration of the activities in the system. Durations are assumed to be known in advance (Schruben 2000).

Each constraint in the model is associated to at least one entity of the simulated system. Thus, it is possible to cluster inequalities using the entities as grouping factor. The first group of constraints ($\mathbf{g}_1$) refer to entity one, the second ($\mathbf{g}_2$) to entity two and so on. Using this group concept, we can rewrite the mathematical programming model as follows:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{g}_1(\mathbf{x}) = \mathbf{0} \\ & \dots \\ & \mathbf{g}_N(\mathbf{x}) = \mathbf{0} \end{aligned} \tag{5}$$

where $\mathbf{g}_i$ is the vector of functions representing the inequalities related to the $i\text{-}th$ entity. Depending on the system, there can be coupling constraints, i.e., there can be constraints containing different entities, especially when finite capacity conditions are present. In this case, $\mathbf{g}_i$ is the vector of function mainly (but not only) related to the i-th entity. In the $G/G/1$ model presented in Section 1 the vector of functions $\mathbf{g}_i$ is represented by equations (2)–(3). Notice that the number of constraints increases as the number of simulated entities does. The number of simulated entities affects also the number of decision variables. Therefore, the complexity of the model is closely related with the length of simulation. As a consequence, as the system becomes more complex, the solution time increases due to the increase in the number of decision variables and constraints. Hence, we can say that for long runs it is not feasible to use mathematical programming for simulation from a computational time point of view.

However, it is possible to notice that the model based on clustered inequalities has an interesting form. Indeed, the problem can be divided into several subproblems, each one representing a portion of the simulation model. More interestingly, this portion is not related to a subsystem of the simulated DES but to a time window of the simulation. This property can be exploited to develop solution methods based on decomposition in submodels. The main benefit of this approach, as described in the following, is that solving the submodels is generally less time consuming than solving the whole mathematical programming model and without loosing in simulation accuracy.

3 METHOD

3.1 Approach

This section describes the time–decomposition approach for solving the mathematical programming model formulated in (5). The approach is based on a simple idea: decomposing the complete model in many simpler submodels that can be solved in a shorter time.
As explained before, the proposed approach can be applied to mathematical programming models for simulation for a specific class of DESs, the main characteristic of which is the use of static rules for dispatching entities and the fixed, and known in advance, sequence of entities moving through the system (Alfieri and Matta 2012a). This class of DESs is large and contains several systems that are relevant in practice such as production lines, supply chains, assembly systems, etc.

Let us decompose the general model (5) in z different submodels. Theoretically, z can be chosen between one (no decomposition) and the number of simulated entities N. Each submodel $s = 1, \dots, z$ simulates a batch with a number b_s of entities. Let Ω_s be the set of entities referring to submodel s. The submodels must simulate all of the entities, i.e., $\bigcup_{s=1}^{z} \Omega_s = \{1, \dots, N\}$. Remember that each submodel s corresponds to a portion of the whole simulation, specifically the part of simulation related to entities in the set Ω_s. Since entity sequence is fixed and cannot be changed, each Ω_s must contain a set of consecutive entities.

Solving each submodel independently is not a good approach because the generic submodel s has to consider the initial status of the system when the first entity in Ω_s enters the system. This initial status depends on the output of the already solved submodels, i.e., on how many entities of the previous submodel are still in the system. This is due to the fact that constraints might contain different entities and hence also the entities considered in the previous submodel. Therefore, the solution of a submodel has to consider the information coming from the previous submodels. This interrelationship is modeled by introducing a new set of constraints built on the basis of the submodel previously solved. For instance, if at the (simulated)

time the first entity of submodel s enters the system, some entities of submodel $s-1$ are still there, some entities in s may be blocked and have to wait for the entities of $s-1$ to leave the system. This new set of constraints is denoted by Υ and its construction depends on the structure of the modeled system.
The mathematical programming formulation for submodel s can be written as:

$$\begin{array}{rc} \min & f_s(\mathbf{x}_s) \\ \text{s.t.} & \mathbf{g}_s(\mathbf{x}_s) = \mathbf{0} \\ & \Upsilon_s = \{\ \mathbf{h}_s(\mathbf{x}_s) = \mathbf{0}\} \end{array}$$

Notice that both the objective function and the constraints refer only to the entities simulated in the submodel. The additional constraint set Υ_s puts into relationship the entities of the current portion of simulation with the entities simulated with the previous submodels; function $\mathbf{h}_s$ is specific for the modeled system. For the first submodel the set Υ_1 is obviously empty if we assume an empty system at (simulated) initial time.

3.2 Time Efforts

Let ϕ_s be the time needed for solving submodel s (*LP time* in the following). This time is a function of the number of entities b_s simulated in the submodel. Indeed, as z increases each submodel becomes simpler because it simulates a smaller number of entities and the LP time should be smaller. Function ϕ_s is assumed random with an unknown distribution. In case of equal number of entities the LP times of the submodels are assumed independent and identically distributed. Also the total time to solve all the submodels (i.e., the sum of all ϕ_s) is a random variable and depends on b_s of each submodel s. As the number of submodels becomes large enough, we know from the central limit theorem that ϕ_T can be approximated by a normal distribution.

In addition to the solution time, each submodel also requires a time φ_s for the input and output of data (*I/O time* in the following). This time is a random variable and depends on the number of constraints Υ_s that have to be added to the submodel s before solving it and on the storage of the solution needed to create the set of constraints Υ_r $(r > s)$ for the next submodels. Notice that the number of constraints Υ_s also depends on the number of entities b_s in submodel s. The total I/O time φ_T, being the sum of the single values φ_s of each submodel, is a random variable. Also in this case, for large enough values of z, function φ_T can be approximated by a normal distribution.

The times for solving the submodels and for manipulating the data can be fitted with standard techniques after having executed some experimentations.
Since each submodel is an LP, time ϕ_s is the time for solving a linear programming problem. Such time is known to be, on the average, a non–linear (polynomial) function of the number of decision variables (Schrijver 1998) that in our problem are related to the number of simulated entities. The shape and order of the LP time function strictly depends on the specific model to be solved.

Computer experiments on specific models in the class of those that can be treated with our approach have shown that a quadratic function can be a good approximation for fitting ϕ_s; some of these experiments will be presented in the next sections. In this special case, the following expression can be used:

$$\phi_s = \beta_0 + \beta_1 b_s + \beta_2 b_s^2 + \varepsilon_\phi, \tag{6}$$

where coefficients β_0, β_1 and β_2 are bounded to be non-negative (the function in the first quadrant must be increasing) and ε_ϕ is the random noise. In general, we can assume that ϕ_s is a monotonically increasing function in b_s. The total LP time ϕ_T can be obtained by summing up the single contributions ϕ_s.

As far as φ_s is concerned, this time strictly depends on the number of data manipulations that are executed to feed Υ_s to the submodel and to store its results. The more the entities in the submodel, the higher the number of constraints in Υ_s and hence the time to feed it and to store the results. However, since the I/O time is, in this case, a "reading and writing" time, the increase can reasonably be assumed linear in b_s, i.e.,

$$\varphi_s = \alpha_0 + \alpha_1 b_s + \varepsilon_\varphi, \tag{7}$$

where coefficients α_0 and α_1 are also bounded to be non-negative and ε_φ is the random noise. The total I/O time φ_T can be obtained by summing up the single contributions φ_s.

Partitioning entities into subsets of different cardinality b_s for each s can be impractical and can make little sense in a real context. Then, in the following, we will consider only the special case in which all submodels have the same number of entities, i.e., $b_s = b\ \forall s = 1,\dots,z$. In this case, if ϕ_s is quadratic and the number of submodels to be solved is $z = N/b$, the total LP time can be expressed as

$$\phi_T = \beta_0 \frac{N}{b} + \beta_1 N + \beta_2 N b + \varepsilon_\phi, \tag{8}$$

and the total I/O time as:

$$\varphi_T = \alpha_1 N + \alpha_0 \frac{N}{b} + \varepsilon_\varphi. \tag{9}$$

3.3 Algorithm to Find the Optimal Number of Submodels

Our aim is to find a partition of the N entities into z subsets so as to have the smallest possible total time (i.e., the sum of LP time and I/O time). This can be achieved finding a value of b that allows to efficiently solve each submodel. It is known that the maximum efficiency (in terms of computational time) is reached for small complexity models, i.e., when the number of variables and constraints is small. In this case, the solution time of an LP model has an almost linear dependence with the number of variables. The reason for this behavior relies in the fact that any polynomial function $f(x)$ changes its slope as the value of x increases and then the first order approximation will be less and less accurate as x becomes larger and larger.

Let us define b_{max} as the value of b such that for any $b \le b_{max}$ function ϕ_s is linear. In this case, any function ϕ_s can be well approximated as:

$$\phi_s = \beta_0 + \beta_1 b + \varepsilon_\phi, \tag{10}$$

and

$$\phi_T = \beta_0 \frac{N}{b} + \beta_1 N + \varepsilon_\phi. \tag{11}$$

Hence, it is necessary to choose $b \le b_{max}$ for having the total LP time linear in N. An estimate of b_{max} can be found by solving the problem for different values of b and testing the adequacy of a linear model. The value b_{max} is the last value of b after which a linear model is no more valid.

However, the batch length b cannot be very small because φ_T is an hyperbola in b and solving z submodels requires more time than solving the complete model. Therefore, the problem reduces to find the smallest possible value for φ_T while ϕ_T is still linear and this happens exactly when $b = b_{max}$.

The overall algorithm to find the optimal solution to the LP model representing the DES dynamic behavior is as follows:

1. **Setting.** Let b_0, δ and r be positive integer parameters and assign them some initial values. Let γ be a non-negative continuous parameter in the interval $(0,1)$ and assign it an initial value.
2. **Initial experiments.** Solve r different submodels (using the equations in Section 3.1) for each batch length $b = b_0 + (m-1)\delta$, with $m = 1,2,3$. Collect the time to solve each LP problem ϕ_s, with $s = 1,\dots,3r$. Set the batch length $b = b_0 + 2\delta$.
3. **Linear fitting.** Assume that the regression linear model (10) fits the data (b,ϕ_s) and test the adequacy of this assumption with the Lack of Fit test at γ confidence level (Montgomery and Peck 1991). If a linear model is valid and the number of customers still to simulate is larger than b, increase the batch length b to $b+\delta$, solve other r different submodels and re–test linearity. If linearity does not fit with data, assign $b_{max} = b - \delta$ and goto step 4.
4. **Simulation.** Solve sequentially one submodel at a time with constant batch length b_{max}.

The algorithm uses at the beginning a three–level experiment for fitting a linear model with the Lack of Fit test. The test is based on partitioning the residual sum of squares into two components, the sum of squares due to pure error and that due to the lack of fit. The relative weight of these two components is then assessed using the Fisher statistic. See the book of Montgomery and Peck (1991) for more details. If the linearity assumption cannot be rejected, a new experiment level is added until the null hypothesis of linearity is rejected. The Lack of Fit tests executed at different iterations are not independent. This dependency increases the probability of stopping the iterations in the linear part because the probability of committing at least one Type I error over all the tests increases iteration by iteration. In alternative, sequential testing procedures could be adopted (Bechhofer, Santner, and Goldsman 1995).

Notice that in steps 2 and 3 no computational time is lost because the r submodels solved for each value of m are not different realization of the same submodel but they are distinct submodels solved one after the other. In such a way the simulation goes on as the submodels are successively solved.

4 G/G/1 QUEUES

4.1 Description

We consider the classical G/G/1 queue with generally distributed random arrival and service times. A number N of customers entering the system and being served is simulated. Customer $i = 1,\dots,N$ arrives at time a_i, waits in the queue if the unique server is busy, and then leaves the system at time y_i when its service is finished. The queue can accommodate an infinite number of customers and the First In First Out (FIFO) rule is adopted.

The LP model of this system is represented by equations (1)–(3) introduced in Section 1. Given customer i, inequalities (2) and (3) together represent the vector of functions $\mathbf{g}_i$. In the following we refer to this model as the *complete* model.

4.2 Decomposed Models

The LP model of the G/G/1 system can be decomposed in a straightforward way since the interdependence between the submodels can easily be captured by simple initial conditions. Let us decompose the complete model in z submodels, each one simulating a set of customers $\Omega_s \subset \Omega$. The customers simulated in submodel s have to follow, according to the FIFO rule, the customers in submodel $s-1$. However, it can happen that a customer of submodel s arrives into the system when some customers of submodel $s-1$ are still there. In this case, it is necessary to introduce an additional constraint imposing that the starting time of the first customer simulated in s has to be greater than or equal to the finishing time of the last customer in submodel $s-1$.

The simulation submodel s can be written as follows:

$$\min \quad \sum_{i\in\Omega_s} y_i \tag{12}$$

$$\text{s.t.} \quad y_i \geq a_i + t_i \qquad i \in \Omega_s \tag{13}$$

$$y_{i+1} - y_i \geq t_{i+1} \quad i \in \Omega_s \tag{14}$$

$$y_f \geq d + t_f \tag{15}$$

where parameter d is set to the departure time of the last customer of submodel $s-1$. The set of the entities simulated in submodel is defined as $\Omega_s = \{\forall i : i = \sum_{j=1}^{s-1} b_j + 1,\dots,\sum_j^s b_j\}$, where b_j is the number of customers simulated in the submodel j. Constraint (15) models the dependence on the previous submodel and allows to apply the proposed decomposition algorithm. Customer $f = \sum_{j=1}^{s-1} b_j + 1$ is the first customer of submodel s. Referring back to Section 3, constraints (13) and (14) represent the vector of functions $\mathbf{g}_s$, related to the entities in Ω_s, while equation (15) is the set Υ_s of initial conditions.

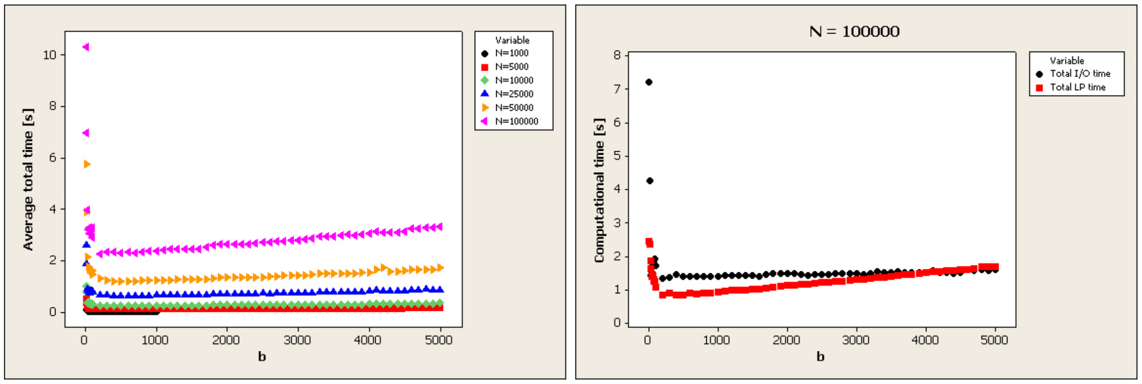

Figure 1: Average total time vs batch length (left); average ϕ_T and φ_T vs batch length for $N = 100,000$ (right).

4.3 Computational Experiments

In this section we first consider a finite horizon simulation of a high traffic intensity single server queue with exponential interarrival and service times. The utilization rate of the queue is $\rho = 1$. This system has been simulated with the complete LP model described in Section 4.1 and the decomposed submodels described in Section 4.2. Linear programs have been solved by using the dual simplex algorithm of IBM ILOG CPLEX.

The computational time for solving the complete model is a non–linear function of the batch length b. Figure 1 on the left reports the average total time (in seconds) as a function of a constant batch length for different levels of N; 100 independent experiments were executed for each point shown in the graph. The batch length that minimizes the total time function is independent from the number of simulated customers N. On the left of the minimum the function assumes high values due to the hyperbolic behavior of the term N/b. On the right, instead, the function increases linearly.

The graph on the right side of Figure 1 shows the detail for $N = 100,000$, plotting the average values of ϕ_T and φ_T as a function of b. The I/O time is hyperbolic for small values of b and then tends asymptotically to a constant value for increasing values of b according to equation (9). The LP time behaves as the total time according to equation (8).

The decomposition algorithm has been applied in a second experiment in which we considered a system with interarrival times uniformly distributed between 1 and 10 time units and service times uniformly distributed between 1 and 5 time units. The algorithm has been executed with the following parameters: $b_0 = 100$, $\delta = 100$, $\gamma = 0.95$ and $r = 50$. These parameters were chosen on the basis of a preliminary design of experiments. Figure 2 shows, on the left, the average total time (in seconds) versus the batch length identified by the algorithm in the experiments with the factor δ at levels 100 and 1000 and factor r at levels 10 and 50. The number of independent replications were 10, and $N = 5,000,000$ and $b_0 = 100$. The chosen values of δ and r (i.e., 100 and 50, respectively) correspond to the points close to the minimum reached for small values of b (Figure 2, left hand side).

Once the parameters b_0, δ, γ and r have been chosen, the algorithm has been applied on 100 independent replications for different values of N and the average total time is shown on the right of Figure 2. Notice that the computational time becomes a linear function of the number of simulated customers when the decomposition algorithm is applied.

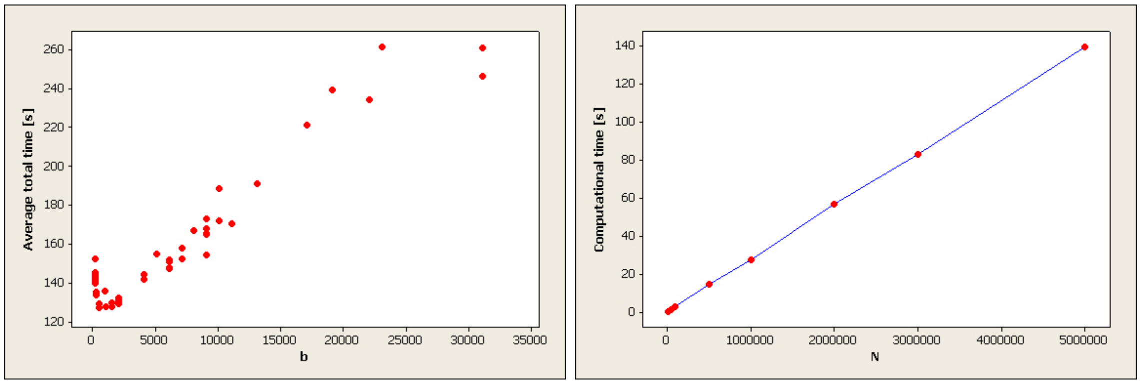

Figure 2: Average total time vs the batch length found by the algorithm (left); average total times vs number of simulated customers (right).

5 TANDEM QUEUING SYSTEMS

5.1 Description

We consider a tandem queuing network with J single–server stages, on which N identical customers have to be processed. The FIFO policy is used to sequence customers in each stage of the line, hence, in practice, no scheduling problem has to be solved. The arrival time a_i of customer i to the line and its processing time t_{ij} at each stage j are assumed known. Due to the FIFO policy, $a_i \leq a_{i+1}$ for each customer i. After having been processed in the first stage, customers proceed to the second stage, then to the third and so forth until they visit the last stage and eventually leave the system. Completion time of customer i at stage j is denoted by y_{ij}.

We consider no balking rate for the system, which implies that the capacity of the first stage (server and queue positions) is infinite. On the contrary, the capacity of stage j $(j >= 2)$ is bounded to be at maximum c_j. If the server in stage j is busy with customer k, customers $i > k$ has to wait in queue at stage j and, due to the finite capacity of the queue, they eventually can block the server in stage $j-1$, $j-2$ and so on. With respect to the blocking phenomenon, the *blocking before service* control rule is assumed for servers (Dallery and Gershwin 1992).

Servers are perfectly reliable and transportation times are considered negligible or already included in service times. Finally, for sake of simplicity, the last stage is never blocked, thus customers completing their processing in the last stage can always leave the system. These assumptions can be easily relaxed and do not limit the generality of the presented results.

This line can be simulated by the following linear programming model, that can be obtained from the formulation of Chan and Schruben (2003):

$$\begin{array}{lll} \min & \sum_{i=1}^{N}\sum_{j=1}^{J} y_{ij} & \quad (16) \\ \text{s.t.} & y_{i1} \geq a_i + t_{i1} \quad \forall i & \quad (17) \\ & y_{i+1,j} - y_{ij} \geq t_{i+1,j} \quad \forall j,\ i = 1,\dots,N-1 & \quad (18) \\ & y_{i,j+1} - y_{ij} \geq t_{i,j+1} \quad \forall i,\ j = 1,\dots,J-1 & \quad (19) \\ & y_{i+c_j,j} - y_{i,j+1} \geq t_{i+c_j,j} \quad i = 1,\dots,N-c_j,\ j = 1,\dots,J-1 & \quad (20) \end{array}$$

The objective function, as in the case of G/G/1 queue, is the minimization of the completion times of each customer in each stage, i.e., the starting times of customers at each stage are not considered since implicitly minimized by the minimization of the completion times.

Equations (17)-(20) describe the system dynamics. In particular, constraints (17) state that parts i can be completed on the first server only if it is arrived at the system and it has been processed. Constraints

(18) and (19) impose that, at the same time, nor a server can process two different customers neither a part can be processed in two different stages, respectively. Constraints (20) prevent a customer from leaving a stage if the immediate downstream queue is full. Finishing times y can assume only positive values in the real domain because the arrival times are non-negative input parameters. The solution of the linear problem provides the optimal values for decision variables y, i.e., the smallest finishing times.

Notice that for each i, equations (17)-(20) correspond to $\mathbf{g}_i$ of Section 2.

5.2 Decomposed Models

The model presented in the previous section can be decomposed into a set of submodels to be sequentially solved. Since the working sequence of customers is known a priori, the decomposition of the problem is equivalent to partitioning the sequence of customers into subsequences. Each subsequence of parts will be independently solved with appropriate initial conditions.

The initial conditions are related to the temporal constraints that link the last customers of each submodel s ($s = 1,\dots,z$) and the first customers of the next submodel to be solved ($s+1$). In fact, if some customers of submodel s are still in process in some stages, the customers in submodel $s+1$ have to wait. In other words, the system at the beginning of a single simulation can be not empty.

To find the number z of submodels, the procedure described in Section 3 can be used as is, since it is problem independent. In this section, instead, we focus on the determination of the initial conditions, i.e., on the determination of Υ.

Assumed already chosen the number b of customers in each submodel, the problem is then to identify which customers of the previous submodel are still in the system (and influence the initial conditions) when the first customer of a given submodel arrives.

Let s be the current submodel to be solved and let I_{prev} be the number of customers of submodel $(s-1)$ that are related to the customers in s. I_{prev} is bounded from above by the total available space in the line (the sum of the maximum queue and server positions of each stage) and from below by 1 (the last customer of the previous submodel). In particular, consider the last customer k of submodel $(s-1)$. When k enters stage j, only customers $i < k$ that are still in queue at stage $j+1$ have to be considered since only they can cause blocking on k thus interfering with the flow of customers of submodel s.

Notice that if the completion time of the last customer of $s-1$ is smaller than the arrival time of the first customer of s, submodels $s-1$ and s are independent and no initial condition for s is necessary.

Given submodel s, I_{prev} can be identified according to the following procedure:

1. Solve the LP represented by submodel $s-1$, obtaining completion time y_{ij} for each customer i in $s-1$ and each stage j;
2. For each customer i in submodel $s-1$, proceeding backward from $i = (s-1)b$, check:
   (a) if $y_{i,j+1} > y_{(s-1)b,j}$ for at least a stage j, customer i is constraining;
   (b) if $y_{i,j+1} < y_{(s-1)b,j}$ for all stages, customer i is non-constraining.

Notice that condition (2.b) can be verified also for a number of customers smaller than the total number that can be accommodated in the system.

Each submodel has then to consider its b customers plus the I_{prev} customers belonging to the previous submodel. Notice that the I_{prev} customers are the first ones of the sequence (due to the FIFO policy) and their service has not to be considered neither in the objective function nor in the constraints since they have already being optimized in the previous submodel.

Using the notation previously introduced, the LP for submodel s $(s = 1,\dots,z)$ is as follows:

$$
\begin{array}{lll}
\min & \sum_{i=1+I_{prev}}^{b+I_{prev}} \sum_{j=1}^{J} y_{ij} & \quad (21)\\
\text{s.t.} & y_{i1} \geq a_i + t_{i1} \quad i = 1+I_{prev},\dots,b+I_{prev} & \quad (22)\\
& y_{i+1,j} - y_{ij} \geq t_{i+1,j} \quad \forall j,\, i = 1+I_{prev},\dots,b+I_{prev}-1 & \quad (23)\\
& y_{i,j+1} - y_{ij} \geq t_{i,j+1} \quad i = 1+I_{prev},\dots,b+I_{prev},\ j = 1,\dots,J-1 & \quad (24)\\
& y_{i+c_j,j} - y_{i,j+1} \geq t_{i+c_j,j} \quad i = 1+I_{prev},\dots,b+I_{prev}-c_j,\, j = 1,\dots,J-1 & \quad (25)\\
& y_{i,j} - d_{i-1,j} \geq t_{i,j}, \quad i = 1+I_{prev},\ j = 1,\dots,J-1 & \quad (26)\\
& y_{i,j-1} - d_{i-c_j,j} \geq t_{i,j-1} \quad i = 1+I_{prev},\dots,c_{j-1}+I_{prev},\ i > c_{j-1},\ j = 2,\dots,J & \quad (27)
\end{array}
$$

The objective function (21) and constraints from (22) to (25) are the same as the complete model, just limited to the customers in the submodel, i.e., they represent the $\mathbf{g}_s$ vector of functions introduced in Section 3. Notice that the last customer in the submodel is customer $b+I_{prev}$, but both the objective function and the constraints do not consider the first I_{prev} customers, since, as explained above, they are customers from the previous submodel, the completion times of which have already been computed and are represented by parameters d_{ij} for the current submodel. In details, parameters d_{ij} are the completion time in stage j of customer i belonging to the previous submodel $(s-1)$. Clearly d_{ij} correspond to the values of variables y in the optimal solution of $(s-1)$.

The Υ constraints containing the initial conditions are represented by equations (26) and (27). In particular, equations (26) allow the first customer of the current submodel (i.e., part $1+I_{prev}$) to be processed in stage j only if stage j has already processed the last customer of the previous submodel (i.e., customer I_{prev}). This condition has to be forced in each stage j. Instead, constraints (27) forbid the first c_{j-1} customers of the current submodel (for each stage j from the second to the last) to leave the stage if the downward buffer is full. If $s=1$ (the first submodel to be solved), no initial condition is necessary, i.e., $I_{prev}=0$ and $\Upsilon_1=\emptyset$.

5.3 Computational Experiments

The decomposition algorithm for the tandem queue has been tested on randomly generated instances with the following characteristics: 4 servers; processing times exponentially distributed with mean equal to 0.4, 0.5, 0.7 and 0.2 time units for the first, second, third and forth server, respectively; customers are all available at time zero (i.e., $a_i=0$ for any i); the buffer capacities are equal to 6, 8 and 5 for stage 2, 3 and 4, respectively. Linear programs have been solved by using the dual simplex algorithm of IBM ILOG CPLEX.

The complete model has been solved for an increasing number of customers N. For each value of N, five replications have been considered. The average times are reported in Figure 3, on the left. It is possible to notice that the I/O time φ_T is linearly increasing with N, since it refers only to the time needed to read the data and write the solution, while the increasing of the LP time ϕ_T (and then of the total time) is non–linear, as discussed in Section 3. Notice however that, for small N, both LP and total time can be well approximated by a first order function.

In the right side of Figure 3 the computational times to solve the sequence of submodels are reported as a function of the number of customers b in each submodel. In this case, we consider a single sample path. It is possible to notice that for small values of b, the I/O time is higher that the LP time and this is reasonable since it refers to data manipulations that are mainly independent from the number of entities in a submodel and hence they are not reducible, while the time to solve the submodel is in the linear region. Increasing b, the number of subproblems to be solved decreases, leading, at the beginning, to a decrease in the I/O time bigger than the increases in the LP time. Increasing b further, the decrease in the I/O time slows down, while the increase in the LP time speeds up. The total time, hence, first decreases and then increases, showing the minimum around 20,000 customers. This point is towards the end of the linear behavior of the LP time, and also corresponds to the steepest decrease in the I/O time.

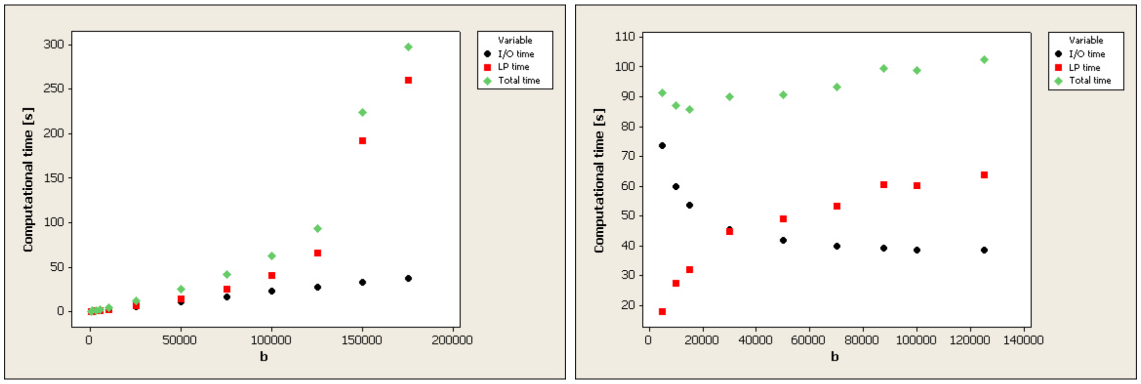

Figure 3: Average LP, I/O and total times for the complete model (left); Average LP, I/O and total time for the decomposed model (right).

6 CONCLUSIONS

In this paper we considered mathematical programming model to simulate DESs. A time–based decomposition algorithm was proposed to overcome one of the major drawback of using mathematical programming to simulate, i.e., the non-linear increasing of computational time as the length of simulation increases.

The developed algorithm first estimates the best number of submodels in which the complete model should be partitioned, and then solves them sequentially, devising, from the solution of each submodel, the initial conditions necessary to solve the next submodel.

Partitioning the complete model into a number of submodels exploits the typical super–additivity of many non–linear functions and makes the mathematical programming approach usable also for long run simulations. In fact, as the number of submodels increases, the number of entities in each submodel decreases, making the computational time of the submodel to decrease as well. However, as the number of submodels increases, the time for data manipulation also increases due to the augmented number of times the solution of a submodel is propagated to the next submodel.

The algorithm estimates the optimal number of submodels solving the trade-off between solution time and data manipulation time. The algorithm has been tested on the G/G/1 queue system. Numerical results confirm that partitioning an LP simulation model allows to execute long runs in a computational time that is a linear function of the number of entities. This important result can extend the use of mathematical programming applied to simulation of DESs. We also applied our algorithm to a tandem queueing system and the results showed that also in this case the reduction in the solution time can be substantial.

When compared with standard simulation, the mathematical programming approach is still looser in terms of computational time. In other experiments (not reported in this paper for reasons of space) we found that the slope of the total computation function of a mathematical programming simulation is steeper than the slope of the standard simulation linear time function. However, having linearized the simulation time will allow to use mathematical programming in a simulation–optimization context in which different models dealing with simulation, or optimization or both have to be used in a unique framework. Notice that in simulation–optimization context standard simulation models are a black–box that differs from optimization models in the mathematics they use and in the software environments they are coded. Having a unique framework is undoubtedly a large advantage for both practitioners and developers.

Finally, the concept of decomposition is valid for LP and MILP problems, however the specific systems for which the proposed algorithm can be fully exploited have still to be clearly defined. A fixed sequence of entities is a prerequisite, but additional features may be requested. All these issues will be the subjects of our research.

REFERENCES

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

Alfieri, A., and A. Matta. 2012b. "Mathematical programming representation of pull controlled single-product serial manufacturing systems". *Journal of Intelligent Manufacturing* 23:23–35.

Bechhofer, R., T. Santner, and D. Goldsman. 1995. *Design and Analysis of Experiments for Statistical Selection, Screening, and Multiple Comparisons*. Wiley Series in Probability and Statistics. Wiley.

Chan, W., and L. Schruben. 2008. "Optimization models of Discrete–Event System Dynamics". *Operations Research* 56 (5): 1218–1237.

Chan, W. K., and L. W. Schruben. 2003, December. "Properties of Discrete Event Systems from their Mathematical Programming Representations". In *Proceedings of the 2003 Winter Simulation Conference*, edited by S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, 496–502. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Dallery, Y., and S. B. Gershwin. 1992. "Manufacturing Flow Line Systems: A Review of Models and Analytical Results". *Queueing Systems Theory and Applications, Special Issue onQueueing Models of Manufacturing Systems* 12 (1-2): 3–94.

Matta, A. 2008, December. "Simulation Optimization with Mathematical Programming Representation Of Discrete Event Systems". In *Proceedings of the 2008 Winter Simulation Conference*, edited by S. J. Mason, R. R. Hill, L. Moench, O. Rose, T. Jefferson, and J. W. Fowler, 1393–1400. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Montgomery, D. C., and E. A. Peck. 1991. *Introduction to linear regression analysis*. John Wiley & sons.

Schrijver, A. 1998. *Theory of Linear and Integer Programming*. John Wiley & sons.

Schruben, L. W. 2000, December. "Mathematical Programming Models of Discrete Event System Dynamics". In *Proceedings of the 2000 Winter Simulation Conference*, edited by J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 381–385. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

ARIANNA ALFIERI is Associate Professor at Politecnico di Torino, where she currently teaches production planning and control and logistic system simulation. Her research area includes scheduling and planning in production and transportation systems. Her email address is arianna.alfieri@polito.it.

ANDREA MATTA is Associate Professor at Politecnico di Milano, where he currently teaches manufacturing and production systems. His research area includes analysis, design and management of production and service systems. His email address is andrea.matta@polimi.it.