Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

SAMPLE-PATH ALGORITHM FOR GLOBAL OPTIMAL SOLUTION OF RESOURCE ALLOCATION IN QUEUEING SYSTEMS WITH PERFORMANCE CONSTRAINTS

Mengyi Zhang
Andrea Matta

Dept. of Mechanical Engineering
Politecnico di Milano
Via La Masa 1
Milan, 20156, ITALY

Arianna Alfieri

Dept. of Management and Production Engineering
Politecnico di Torino
Corso Duca degli Abruzzi 24
Turin, 10129, ITALY

ABSTRACT

Resource allocation problems with performance constraints (RAP–PC) are a category of optimization problems on queueing system design, widely existing in operations management of manufacturing and service systems. RAP–PC aims at finding the system with minimum cost while guaranteeing target performance, which usually can be obtained only by simulation due to complexity of practical systems. This work considers the optimization of the integer–ordered variables, which the system performance is monotonic on. It proposes an algorithm providing a sample–path exact solution within finite time. Specifically, the algorithm works on the mathematical programming model of RAP–PC and uses logic–based exact and gradient–based approximate feasibility cuts to define and reduce the feasible region. Results on randomly generated instances show that the proposed approach can solve at optimality up to 9–dimension problems within two hours and feasible good quality solutions can be found faster than the state–of–the–art algorithm.

1 INTRODUCTION

The resource allocation problem (RAP) of queueing systems represents a category of optimization problems in queueing system design, which deals with the decision about the amount of resources, such as servers and queue capacity, to allocate to each stage. It generalizes the well–known buffer allocation problem and server allocation problem. Exactly as in the mentioned problems, which RAP is a generalization of, the main aspect, from an operational perspective, is the trade–off between the system performance and the overall cost. Roughly speaking, increasing the quantity of the allocated resources allows to reach a higher performance, but, on the other hand, it will also increase the cost. RAPs can be categorized depending on whether the performance and/or the cost is managed through objective function or constraints. This work considers, specifically, the RAP with performance constraints (RAP–PC) in queueing systems. In the studied problem, once a set of performance indicators have been identified, a target is set to each of them as constraint, and the objective is to achieve the minimum total cost. Applications of RAP–PC can be widely found in the field of manufacturing and service system operations management, such as call center staffing, buffer allocation, emergency room staffing, etc.

As queueing systems representing practical settings are usually subject to blocking, dispatching policies and non-Markov property, discrete event simulation (DES) is one of the most used tools for performance evaluation, and, hence, simulation–optimization algorithms have to be used for RAP–PC. More specifically, the relevant research field is discrete optimization via simulation with stochastic constraints, since resources are represented by integer–ordered variable and the feasibility of a system design can only be identified after simulation. This work proposes a sample average approximation (SAA) algorithm, which finds a global optimal solution for RAP–PC of given sample paths within finite time. Once applying SAA, a stochastically constrained problem becomes a deterministic problem, but still with unknown feasible region.

Several approaches has been proposed to solve constrained problems. In sampling–based algorithms, both globally convergent approaches (e.g., nested partition (Shi and Ólafsson 2000) and probabilistic branch and bound (Norkin et al. 1998)) and locally convergent approaches (e.g., COMPASS (Hong and Nelson 2006) and adaptive hyperbox approximation (Xu et al. 2013)), the unknown feasible region makes it troublesome to efficiently partition the areas and, hence, to efficiently compute promising solutions. In this case, a penalty–type method could be applied together with the sampling–based methods, leading to a sequence of unconstrained problems that optimize the sum of the objective function and of the penalty for constraint violation. However, as the penalty parameter is usually iteratively tuned and the optimization model keeps being changed, which makes the approach inefficient from a computational point of view. Gradient–based approaches, such as R-SPLINE (Wang et al. 2013), can handle the unknown feasibility, but are only locally convergent.

The approach for the solution of RAP–PC proposed in this work, includes features that make it able to provide the global optima of the SAA model within finite time and avoid the use of intensive simulation budget for searching for global optima. The algorithm works on the mathematical programming (MP) model of RAP–PC and its original contribution is the use of exact and approximate feasibility cuts to define and reduce the feasible region. Specifically, using gradient to generate approximate cut has never been investigated in literature. The gradient estimation on the integer variable, i.e., the resource capacity, is enabled by exploring the structure of the DES through its mathematical programming representations (MPR) (Chan and Schruben 2008). This gradient estimation approach takes the simulation experiment of one design point only, which provides the algorithm high efficiency in finding good solution within finite time. The exact cuts are logic constraints, which are formulated based on the property of the optimization problem. Thus, the approach proposed in this work considers neither the simulation model nor the optimization problem as purely black–box.

The reminder of the paper is organized as follows. In Section 2, RAP–PC is formally defined, together with the approach to identify a so–called *resource–type* variable through the MPR of DES. The algorithm and the cut formulation are presented in Section 3. Numerical analysis is discussed in Section 4, while Section 5 gives concluding remarks and future research directions.

2 PROBLEM DEFINITION

In this paper, a system composed of J multi–resource *stages* is considered. All the resources in one stage are identical and working in parallel, and resources in different stages can differ from each other; the requirement to be identical implies that a stage cannot be considered as a monolithic composition of both servers and queues. The capacity of each stage j, i.e., the amount of parallel resources in it, is the decision variable of RAP–PC, denoted by x_j, and the vector $\mathbf{x} = [x_1, x_2, ..., x_J]^T$ is the compact representation of the set of all the decision variables. The initial search area $\mathbb{X}$ is considered as a box–shape subset of integer–ordered lattice of dimension J, i.e., each variable x_j is bounded by a lower bound a_j and an upper bound b_j. The set $\mathbb{X}$ should be well defined to avoid having unstable systems, which is usually not tough. Using vector $\mathbf{c}$ to collect in a compact form the cost of each single resource, RAP–PC can be mathematically formulated as follows:

$$RAP-PC: \quad \min_{\mathbf{x}\in\mathbb{X}} \{\mathbf{c}^T\mathbf{x}\}$$
$$s.t.\ h_l(\mathbf{x}) \le p_l^* \quad \forall\ l = 1, ..., L \tag{1}$$

The left hand side of constraints (1) represents the performance of the system with resource capacity $\mathbf{x}$, and the right hand side is the target performance. Multiple performance constraints (indexed by l) can be handled.

The variable vector $\mathbf{x}$ is not a generic integer–ordered variable vector; instead, a variable is defined as *resource–type* if the system performance is monotonic on it. They are formally defined as follows:

Definition 1 An integer–ordered variable is a *resource–type* variable, if and only if the system performance will not be worsened by increasing its value, assuming that all the other variables representing the system design are unchanged. Formally, a variable x_j is a resource–type variable if and only if the following inequality holds:

$$h_l(\mathbf{x}) \geq h_l(\mathbf{x}+\hat{\mathbf{e}}_j), \;\; \forall \mathbf{x}, \, \mathbf{x}+\hat{\mathbf{e}}_j \in \mathbb{X}, \, l=1,...,L$$

where $\hat{\mathbf{e}}_j$ is a vector of dimension J, its j-th entry is equal to one, and all the other entries are equal to zero.

According to Definition 1, if $\bar{\mathbf{x}}$ is a feasible solution satisfying constraints (1), then any $\mathbf{x}$ greater than or equal to $\bar{\mathbf{x}}$ is feasible. If $\bar{\mathbf{x}}$ is an infeasible solution violating constraints (1), then any $\mathbf{x}$ smaller than or equal to $\bar{\mathbf{x}}$ is infeasible. (Vector $\mathbf{x}$ is defined to be greater than or equal to $\bar{\mathbf{x}}$ if every element x_j of $\mathbf{x}$ is greater than or equal to the same element $\bar{x}_j$ in $\bar{\mathbf{x}}$.) Thus, the feasible region of RAP–PC has a stair shape boundary on lower left side, and multiple local optima exist. Resource–type variables widely exist in manufacturing and service systems (Buzacott and Shanthikumar 1993). For instance, the buffer and server capacity in multi-stage serial, assembly/disassembly, split/merge systems are resource–type.

3 FINDING THE GLOBAL OPTIMA OF THE SAA MODEL

The proposed algorithm is based on a mixed integer programming (MIP) model of RAP–PC (denoted by RAP–PC–MIP) in which the feasible region is defined through *feasibility cuts*. Since the feasible region is unknown at the beginning, RAP–PC–MIP is initialized with $\mathbb{X}$; then, the feasibility cuts are gradually generated and added when infeasible solutions are visited, and the solution to be simulated in the next iteration is obtained by solving the MIP. For each infeasible solution, two types of cuts, approximate and exact, can be developed from simulation experiments. The approximate cut is based on the gradient of the performance, while the exact cut is a logic constraint based on the definition of resource–type variables. Using approximate cuts usually leads to defining as infeasible a large set of solutions especially when the solution under study is far from the boundary, thus it largely reduces the feasible region; however, it causes the risk of recognizing the optima as infeasible, and then missing optimality. Using exact cuts, instead, is inefficient, since the partition is very conservative in defining infeasible solutions. To efficiently search for the global optimum, the proposed algorithm uses both approximate cuts and exact cuts. Specifically, the approximate cuts are used in RAP–PC–MIP allowing the search to approaching the boundary and finding feasible solutions at the early time of the solution process, which are also candidates of incumbent solutions. Each time a feasible solution is found, approximate cuts are replaced by exact cuts, and the lower bound of the optimum could be then updated. The procedure stops when the gap between the incumbent solution and the lower bound reaches zero (or is considered negligible because it is below a given threshold).

In the next sections, approximate cuts and exact cuts are presented, followed by the pseudo code of the overall solution algorithm.

3.1 Gradient–Based Approximate Cut

In this section, for sake of simplicity, a single performance measure is considered (i.e., the index l of performance constraints is dropped), and the variable $\mathbf{x}$ is extended to be real-valued. For an infeasible solution $\bar{\mathbf{x}}$ violating the performance constraint $h(\mathbf{x}) \leq p^*$, the gradient of $h(\mathbf{x})$ at $\bar{\mathbf{x}}$ is denoted by $\boldsymbol{\lambda}(\bar{\mathbf{x}})$ and the following approximate cut $CA(\bar{\mathbf{x}})$ is defined:

$$CA(\bar{\mathbf{x}}): \; \boldsymbol{\lambda}^T(\bar{\mathbf{x}})(\mathbf{x}-\bar{\mathbf{x}})+h(\bar{\mathbf{x}}) \leq p^*. \tag{2}$$

Equation (2) has the same form as the generalized Benders feasibility cut (Geoffrion 1972). The approximation of the cut (2) arises from the fact that the generalized Benders decomposition framework is developed under the assumption of $h(\mathbf{x})$ being convex, while the convexity for $h(\mathbf{x})$ is not assumed in this work.

Generally speaking, any gradient estimator can be used in equation (2). However, considering the complexity of generic queueing systems, constructing an estimation of $\boldsymbol{\lambda}(\bar{\mathbf{x}})$ is not easy. In this work, an approach to approximately estimating the gradient vector $\boldsymbol{\lambda}(\bar{\mathbf{x}})$ for resource–type variables $\mathbf{x}$ in generic queueing systems using simulation runs of only one design point is proposed. The gradient estimation is based on an equivalent linear programming model (LPM) describing the dynamics in the DES trajectory with fixed sample paths (i.e., whose solution provides the same result as launching the simulator (Chan and Schruben 2008)). In the context of linear programming, sensitivity analysis can direct the gradient estimation, which involves the dual solution, i.e., Lagrangian multipliers, of the LPM.

The LPM representation of the simulation run of the queueing system is developed using the modeling framework presented in Chan and Schruben (2008). Even though Chan and Schruben (2008) proposed the LPM of single-replicate simulation only, extending the formulation to suit multiple-replicate simulation is trivial. The LPM of K replicates is introduced in the following.

$$\min \qquad h(\bar{\mathbf{x}}) = \frac{1}{K}\sum_{k=1}^{K}\sum_{(\xi,i,j)\in\mathscr{E}} f_{i,j}^{\xi} e_{i,j,k}^{\xi} \tag{3}$$

$$s.t. \qquad e_{i',j',k}^{\xi'} - e_{i,j,k}^{\xi} \geq t_{i,j,i',j',k}^{\xi,\xi'} : \; u_{i,j,i',j',k}^{\xi,\xi'}$$

$$\forall\ (\xi,i,j),(\xi',i',j') \in \mathscr{E}/\{(\xi,i,j),(\xi',i',j')\} \in \mathscr{P},\ k=1,...,K \tag{4}$$

$$e_{i,j,k}^{\xi} \in \mathbb{R}^{+} \quad \forall\ (\xi,i,j) \in \mathscr{E},\ k=1,...,K$$

The real-valued variables $e_{i,j,k}^{\xi}$ represent the time of event execution; specifically, it refers to the i-th execution of event type ξ in stage j in the k-th replicate. The objective function (3) represents the performance measures $h(\bar{\mathbf{x}})$ as linear combination of event occurring times (e.g., the average system time of replicate k is written as $\frac{1}{I}\sum_i(e_{i,J,k}^{departure} - e_{i,1,k}^{arrival})$, where $e_{i,J,k}^{departure}$ is the time of departure from the last stage, and $e_{i,1,k}^{arrival}$ is the time of arrival at the system, and I is the total number of parts). Constraints (4) represent the triggering relationships between event $e_{i,j,k}^{\xi}$ and $e_{i',j',k}^{\xi}$. The left hand side states that after the execution of $e_{i,j,k}^{\xi}$, it is possible to add event $e_{i',j',k}^{\xi'}$ into the future event list. For instance, $e_{i,j,k}^{arrival} - e_{i,j-1,k}^{departure}$ states that the *arrival* at stage j must happen after the *departure* from upstream stage $(j-1)$. The right hand side is equal to the lag between the two event occurrences. For instance, $t_{i,j-1,i,j,k}^{departure,\ arrival} = 0$ shows that once a job leaves station $(j-1)$, it can enter station j immediately. The dual variables related to the constraints are denoted by $u_{i,j,i',j',k}^{\xi,\xi'}$. $\mathscr{E}$ represent the set of all possible combinations of event type, occurrence and stage, and $\mathscr{P}$ represents the set of all the possible pairs of triggering and triggered events, respectively. Not all the possible constraints (4) are a–priori added to the model, but the needed ones are added when the events realize, in a next–event time advanced mechanism framework.

The LPM is generated automatically from an event–based DES model during its realization, hence, it is not constructed as a *simulator*, but as a record of simulation history. An event–scheduling simulator is executed in the following way: the next event $e_{i,j,k}^{\xi}$ (i.e., the event with the earliest time in the future) occurs and the global clock is advanced to its occurring time $\bar{e}_{i,j,k}^{\xi}$, the event occurrence changes the state of the system, and the state change enables to add new events $e_{i',j',k}^{\xi'}$ to the event list together with their occurring time $\bar{e}_{i',j',k}^{\xi'}$. Thus, a triggering relationship between event $e_{i,j,k}^{\xi}$ and event $e_{i',j',k}^{\xi'}$ is identified, since the occurrence of event $e_{i,j,k}^{\xi}$ schedules event $e_{i',j',k}^{\xi'}$, and the left hand side of a constraint is written as $e_{i',j',k}^{\xi'} - e_{i,j,k}^{\xi}$. As for the right hand side, $t_{i,j,i',j',k}^{\xi,\xi'}$ is replaced by $\bar{e}_{i',j',k}^{\xi'} - \bar{e}_{i,j,k}^{\xi}$ representing the lag between the two actual occurring times, where $\bar{e}_{i,j,k}^{\xi}$ is a number representing the actual occurring time, while $e_{i,j,k}^{\xi}$ represents a variable. All the triggering relationships are transformed into constraints (4) in the same way.

Even though, the event occurring times, i.e., the primal solution of the LPM, can be easily collected during the simulation run as $e^{\xi}_{i,j,k} = \bar{e}^{\xi}_{i,j,k}$, meaning that the LPM is never solved with mathematical programming methods, obtaining the dual solution $u^{\xi,\xi'}_{i,j,i',j',k}$ is not as easy as primal solution. However, also the dual solution can be obtained after a simulation run as follows. The LPM has an equivalent graph representation, which is a directed spanning tree with nodes representing events and arcs representing triggering relationships, the so–called event relationship graph (ERG). The dual of the LPM is a maximum flow problem on the spanning tree (Chan and Schruben 2008), and the dual variables $u^{\xi,\xi'}_{i,j,i',j',k}$ are the flows carried by the arc from $e^{\xi}_{i,j,k}$ to $e^{\xi'}_{i',j',k}$. The coefficient $\frac{1}{K}f^{\xi}_{i,j,k}$ constrains the flow balance of node $e^{\xi}_{i,j,k}$. If $f^{\xi}_{i,j,k}$ is positive, the node is a sink, while if it is negative, the node is a source. The dual solution can be obtained by finding the path from each sink to the root node of the spanning tree, which is unique for each sink, and calculating the flow of each arc as the sum of all the flows passing by.

Knowing the LPM and its dual solution, the gradient can be calculated. In fact, among all the constraints (4), there are some that are particularly relevant to RAP–PC, which are specifically

$$e^{arrival}_{i,j,k} - e^{departure}_{i-\bar{x}_j,j,k} \geq 0: \; u^{arrival,\,departure}_{i,j,i-\bar{x}_j,j,k} \quad \forall i,j,k. \tag{5}$$

Constraints (5) state the fact that the i-th arrival cannot be earlier than the $i-\bar{x}_j$ departure in the same resource stage j, due to the blocking caused by the limited capacity $\bar{x}_j$ of the resource. If a perturbation $\Delta\bar{x}_j$ is applied to $\bar{x}_j$, and it introduces the perturbation $\delta_j(\bar{x}_j)$ on the right hand side of equation (5), and the dual optimal solution is known to be $\bar{u}^{arrival,\,departure}_{i,j,i-\bar{x}_j,j,k}$, based on the sensitivity analysis of linear programming, the j-th component of the gradient vector $\lambda_j(\bar{\mathbf{x}})$ can be estimated as:

$$\lambda_j(\bar{\mathbf{x}}) = \frac{\delta_j(\bar{x}_j)}{\Delta\bar{x}_j}\sum_{k=1}^{K}\sum_{i}\bar{u}^{arrival,\,departure}_{i,j,i-\bar{x}_j,j,k}.$$

If the capacity $\bar{x}_j$ is increased by one, i.e., $\Delta\bar{x}_j = 1$, the left hand side of equation (5) becomes $e^{arrival}_{i,j} - e^{departure}_{i-(\bar{x}_j),j} = e^{arrival}_{i,j} - e^{departure}_{i-(\bar{x}_j+1),j} + (e^{departure}_{i-(\bar{x}_j+1),j} - e^{departure}_{i-(\bar{x}_j),j}) \geq e^{departure}_{i-(\bar{x}_j+1),j} - e^{departure}_{i-(\bar{x}_j),j}$, showing that the perturbation to the right hand side $\delta_j(\bar{x}_j)$ can be approximated by the negative inter–departure time. Thus, $\frac{\delta_j(\bar{x}_j)}{\Delta\bar{x}_j}$ can be set to the negative inter–departure time of stage j, denoted by $-\tau_j$, and the j-th component of the gradient vector $\boldsymbol{\lambda}(\bar{\mathbf{x}})$ is as follows:

$$\lambda_j(\bar{\mathbf{x}}) = -\tau_j\sum_{k=1}^{K}\sum_{i}\bar{u}^{arrival,\,departure}_{i,j,i-\bar{x}_j,j,k}.$$

3.2 Exact Cut

The exact cut is based on the definition of resource–type variables- Before dealing with the cut generation, the concept of dominance of one solution $\bar{\mathbf{x}}_1$ to another solution $\bar{\mathbf{x}}_2$ has to be defined. If the vector of difference between the two solutions $\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2$ has only non–negative entries, then $\bar{\mathbf{x}}_1$ dominates $\bar{\mathbf{x}}_2$. According to Definition 1, $h(\bar{\mathbf{x}}_1) \leq h(\bar{\mathbf{x}}_2)$, thus if $\bar{\mathbf{x}}_1$ violates some performance constraint, $\bar{\mathbf{x}}_2$ will also violate the same constraint. Therefore, an exact cut at the infeasible solution $\bar{\mathbf{x}}$ partitions the feasible region $\mathbb{X}$ into two sets, an infeasible set including all the solutions $\bar{\mathbf{x}} - \mathbf{x}'$, where $\mathbf{x}'$ has non–negative entries only, and the complementary set. To formulate the partition with integer linear constraints, a set of binary variables $y_{j,k} \in \{0,1\}$ are introduced as follows:

$$y_{j,k} = \begin{cases} 1, & \forall\, k \leq x_j \\ 0, & \forall\, k \geq x_j + 1, \end{cases}$$

guaranteed by the following constraints:

$$y_{j,k} \le y_{j,k-1} \; \forall j = 1,...,J, \; k = a_j+1,...,b_j$$
$$x_j = a_j + \sum_{k=a_j+1}^{b_j} y_{j,k} \; \forall j = 1,...,J,$$

where a_j and b_j are the lower bound and the upper bound on the value of x_j, respectively. The exact cut $CE(\bar{\mathbf{x}})$ is then formulated as:

$$CE(\bar{\mathbf{x}}): \sum_{j:\; \bar{x}_j < b_j} y_{j,\bar{x}_j+1} \ge 1.$$

The dominance between exact cuts $CE(\bar{\mathbf{x}}_1)$ and $CE(\bar{\mathbf{x}}_2)$ exists if $\bar{\mathbf{x}}_1$ dominates $\bar{\mathbf{x}}_2$, since the infeasible set partitioned by $CE(\bar{\mathbf{x}}_2)$ is a subset of the infeasible set partitioned by $CE(\bar{\mathbf{x}}_1)$. Thus, finding dominating infeasible solutions (DIS) can define larger infeasible region with fewer cuts, which improves the computation efficiency.

A heuristic to find DIS based on the approximate gradient $\boldsymbol{\lambda}(\bar{\mathbf{x}})$ is here proposed. In the gradient vector $\boldsymbol{\lambda}(\bar{\mathbf{x}})$, if the absolute value of one or several elements are significantly greater than the other elements satisfying the equality $\frac{\lambda_j(\bar{\mathbf{x}})}{\lambda_{max}(\bar{\mathbf{x}})} \ge \alpha$, where $\lambda_{max}(\bar{\mathbf{x}})$ denotes the maximum element of $\boldsymbol{\lambda}(\bar{\mathbf{x}})$ and α is a user–defined threshold, then the performance is more impacted by the resource capacity limitation of such stages, and they can be regarded as the bottlenecks of the system. A dominating system $\bar{\mathbf{x}}_d(\bar{\mathbf{x}}) = [\bar{x}_{d,1},...,\bar{x}_{d,J}]^T$ of infeasible solution $\bar{\mathbf{x}}$ can be constructed by setting the resource capacity of all the non–bottleneck stages to the upper bound and keeping the capacity of the bottlenecks unchanged, i.e.,

$$\bar{x}_{d,j} = \begin{cases} \bar{x}_j, & \forall\; j = 1,...,J \; such \; that \; \frac{\lambda_j(\bar{\mathbf{x}})}{\lambda_{max}(\bar{\mathbf{x}})} \ge \alpha, \\ b_j, & \forall\; j = 1,...,J \; such \; that \; \frac{\lambda_j(\bar{\mathbf{x}})}{\lambda_{max}(\bar{\mathbf{x}})} < \alpha. \end{cases}$$

The dominating system is then simulated, and, if it is infeasible, a tighter exact cut $CE(\bar{\mathbf{x}}_d(\bar{\mathbf{x}}))$ can be generated and the cut of the original system $CE(\bar{\mathbf{x}})$ is replaced by $CE(\bar{\mathbf{x}}_d(\bar{\mathbf{x}}))$; otherwise, if it is feasible, the DIS will be the original system itself, i.e., $\bar{\mathbf{x}}_d(\bar{\mathbf{x}}) = \bar{\mathbf{x}}$.

Figure 1 shows the partition made by the approximate cut $CA(\bar{\mathbf{x}})$, the original exact cut $CE(\bar{\mathbf{x}})$, and the dominating exact cut $CE(\bar{\mathbf{x}}_d(\bar{\mathbf{x}}))$ with $\bar{\mathbf{x}} = [1,4]^T$ in a two–dimension problem. The approximate cut has a linear shape in the two-dimension space, and its slope shows the gradient of the performance, while the left side of the line is the infeasible set. Some feasible solutions are misleadingly assigned to the infeasible area. The infeasible set defined by the exact cut at $[1,4]$, i.e., the left lower corner of the line, includes only solutions dominated by $[1,4]$, which is a very small set. The slope of the approximate cut shows that the system performance is more sensitive to x_1, thus stage 1 is the bottleneck of the system. The dominating system is constructed as $[1,8]$, by setting x_2 to upper bound 8 and keeping x_1 unchanged. After simulating $[1,8]$, it is found infeasible, so it is a DIS of $[1,4]$. An exact cut at $[1,8]$ is then generated, and it defines a larger infeasible set compared with the original exact cut, but it does not exclude feasible solutions from the future search area.

3.3 Algorithm

The complete algorithm for solving RAP–PC is summarized in Algorithm 1. The resource capacities are initialized to the lower bound. The searching region of RAP–PC–MIP is initialized to $\mathbb{X}$, and the lower and upper bounds of the objective function, C^L and C^U, respectively, are set considering the upper bound and lower bound of the capacity of each stage. Lines 7 to 11 show that approximate cuts are generated and used in the model when infeasible solutions are found. Once a feasible solution is visited, the upper

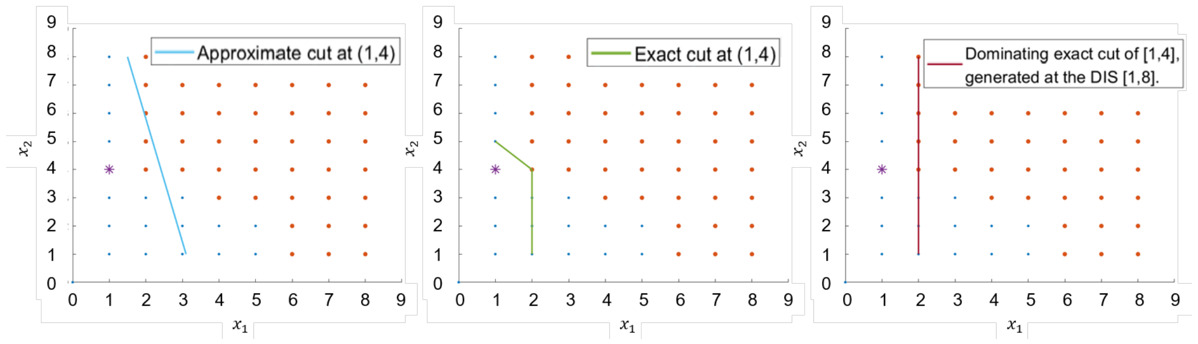

Figure 1: Approximate, exact, and dominating exact cuts.

bound C^U, which is also the candidate of incumbent solution, can be updated after comparing the value of the found feasible solution and that of the current incumbent. Then, as shown in lines 13 to 18, all the currently used approximate cuts are replaced by exact cuts of the DIS. If there are only exact cuts in RAP–PC–MIP, the solution is the new lower bound C^L. The algorithm terminates when the gap between the upper bound and lower bound is within a tolerance or the time limit is exceeded.

4 NUMERICAL ANALYSIS

The proposed approach is applied to the server allocation problem of serial–parallel queueing system, aiming at finding the server number in each stage that allows to achieve the minimum cost while guaranteeing that the average system time over all the jobs does not exceed a target value. As each stage cannot be considered a monolithic entity of buffer and servers, the serial–parallel queueing system has to be seen as composed by a sequence of stages of servers s_j alternated by stages of buffers b_j (Figure 2). Jobs arrive at the first stage of the system, which is a buffer with infinite capacity, with a general arrival process. In each buffer stage, which has a finite given capacity (with the exception of the first one), jobs are managed by a first–in–first–out policy. In a stage of servers, the first finished job is the first to be released to the next stage, which is a stage of buffer. After being processed by the last server stage, each job immediately leaves the system.

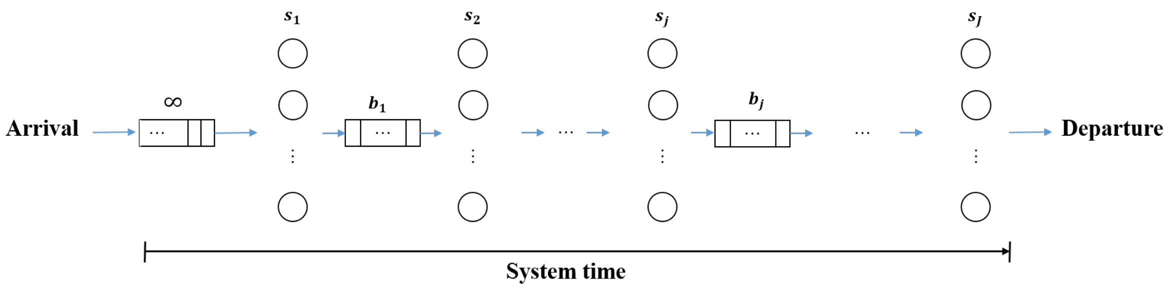

Figure 2: Serial–parallel queueing system.

The algorithms are implemented in Java and Cplex 12.10 is used for solving the master problem. The experiments are conducted on a cluster of DELL M630 with Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz processors and 256 Gb RAM. Cplex can use up to 16 threads.

Two experiment settings, namely ‘Norm’ and ‘Exp’, have been used and are listed in Table 1. Stage number J is varied from ‘min J’ to ‘max J’, where ‘max J’ represents the maximum number of stages for which the problem can be solved within 2 hour time limit. Five replicates have been run for each parameter

Algorithm 1 MIP–based algorithm.

```
Input:
    Lower bound $\mathbf{a} = [a_1,...,a_J]$ and upper bound $\mathbf{b} = [b_1,...,b_J]$ of resource capacity $\mathbf{x}$, such that $a_j \leq x_j \leq b_j \; \forall \; j = 1,...,J$.
    Number of replicates $K$, and the sample paths of the $K$ replicates.
    Tolerance of optimality gap $\varepsilon_{opt}$.
    Optional input: time limit of the algorithm $T_{lim}$.
Ensure:
    Sample-path global optimal $\mathbf{x}^*$.
 1: Initialize system with lower bound $\mathbf{x} \leftarrow \mathbf{a}$
 2: Initialize incumbent with upper bound $\mathbf{x}^* \leftarrow \mathbf{b}$.
 3: Initialize lower bound of the objective $C^L \leftarrow \mathbf{c}^T\mathbf{a}$.
 4: Initialize upper bound of the objective $C^U \leftarrow \mathbf{c}^T\mathbf{b}$.
 5: Add initial constraints which defines $\mathbb{X}$ to the RAP–PC–MIP.
 6: while $C^U - C^L > \varepsilon_{opt}$ and $T_{lim}$ is not exceeded do
 7:     while There exists at least one violated performance constraint do
 8:         Generate one approximate cut $CA(\bar{\mathbf{x}}, l)$ for each violated constraints $l$ and add all the generated cuts to the RAP–PC–MIP.
 9:         $\bar{\mathbf{x}} \leftarrow$ solution of the RAP–PC–MIP.
10:         Simulate the system of $\bar{\mathbf{x}}$.
11:     end while
12:     Update upper bound $C^U \leftarrow \min\{\mathbf{c}^T\bar{\mathbf{x}},\ C^U\}$. If $\mathbf{c}^T\bar{\mathbf{x}} < C^U$, then $\mathbf{x}^* \leftarrow \bar{\mathbf{x}}$
13:     if There exist approximate cuts in RAP–PC–MIP then
14:         For all the currently used approximate cuts $CA(\bar{\mathbf{x}}^r, l)$, find dominating infeasible solution $\bar{\mathbf{x}}_d(\bar{\mathbf{x}}^r)$ and replace approximate cuts $CA(\bar{\mathbf{x}}^r, l)$ by exact cuts $CE(\bar{\mathbf{x}}_d(\bar{\mathbf{x}}^r), l)$ of the DIS.
15:         $\bar{\mathbf{x}} \leftarrow$ solution of the RAP–PC–MIP.
16:         Simulate the system of $\bar{\mathbf{x}}$.
17:         Update lower bound $C^L \leftarrow \max\{\mathbf{c}^T\bar{\mathbf{x}},\ C^L\}$.
18:     end if
19: end while
```

combination. The threshold α, used for identifying the DIS from the gradient, has been set to 0.3 in all the cases. Number of replicates is set to be 1.

Since the proposed method involves the solution of the MIP model, the optimization time, i.e., the total time spent solving the MIP in all the iterations, can be the most of the computation time in high dimensional cases. For instance, for Exp experiment with J equal to 8, 63.0% of the computation time is spent for optimization on average. Thus, analyzing the computation time, i.e., the sum of simulation time and optimization time, is very relevant. The maximum number of stages that can be solved to optimality within 2 hour is 8 and 9 for Exp and Norm, respectively. Figure 3 shows the relationship between the computation time and the dimension J of the instances for both Exp and Norm experiment. According to the fitted regression model, the total computation time is quadratic exponential on J.

A good feasible solution can be found in early time. Figure 4 shows the convergence of the upper and lower bound of the optimum for a single experiment of the 8–stage Exp experiment and the 9–stage Norm experiment. In the 8–stage Exp experiment, the algorithm terminates in 2358.8 seconds with an optimal solution equal to 62, but a feasible solution equal to 62 has been visited at time 377.0 second, and the first feasible solution equal to 65 is found at time 0.668 second, after having visited only 6 solutions. Similarly for the 9–stage Norm experiment, the algorithm terminates in 5025.5 seconds with an optimal solution equal to 63, but a feasible solution equal to 63 has been visited at time 1.0 second, after having visited

	Norm	Exp
Inter-arrival time distribution	$N(2,2)$ truncated	$Exp(2)$
Service time distribution	$N(10,10)$ truncated	$Exp(10)$
Target system time	$11.2J$	$11.2J$
Inter–stage buffer capacity	5	5
Server number lower bound	6	6
Server number upper bound	15	15
Server cost	1	1
Simulation length (number of jobs)	$100,000$	$100,000$
min J	3	3
max J	9	8

Table 1: Experiment settings

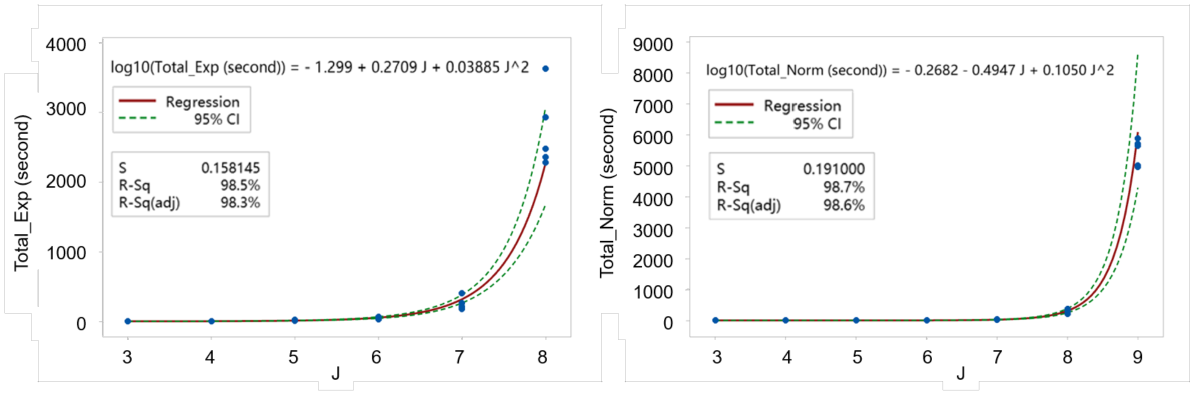

Figure 3: Computation time of Exp and Norm experiment. (Time limit equal to 7200 seconds)

only 9 solutions. The early good performance can be achieved since the proposed gradient estimation approach needs to evaluate only one design point through simulation. Moreover, since the global optimum is guaranteed by the exact cuts, the risk of getting a bad result by using approximate gradients is wiped out after the exact logic–based cuts replace the gradient–based cuts.

The performance of the proposed approach is also compared to the adaptive hyperbox algorithm (AHA) (Xu et al. 2013) combined with penalty–type framework proposed in Park and Kim (2015). AHA is designed for non–constrained high–dimensional discrete optimization problems where the evaluation is obtained via black–box simulation. Park and Kim (2015) proposed certain sequences of penalty, so that non-constrained simulation-optimization algorithms can be adapted to solve constrained problems. As for the benchmark algorithm, the penalty sequence 1 (PS1) proposed in Park and Kim (2015) has been used, and the parameters are set as recommended. As for AHA, the number of sampled solutions in each iteration is varied among 2, 5, 10. The benchmark algorithm is denoted by AHA+PS1, and the proposed approach is denoted by DEO. Since the sample path is fixed, repeatedly evaluating the same solution is not necessary, and the performance is stored and read in the memory for all the already visited solutions. A 20–stage Norm case has been solved on five independent sample paths. Considering the noise of sampling in AHA, AHA+PS1 is launched five times for each sample path. Figure 5 shows the empirical convergence of the incumbent over computation time for the proposed approach and AHA+PS1. The computation time for the proposed approach takes into account the time for simulation and time for solving MILP. Figure 5 shows that the AHA+PS1 can find a feasible solution in a short time, because, under the experimental settings, the feasible region is large and, hence, it is easy to visit a feasible solution with random sampling. However, ever since the first feasible solution is visited by the proposed approach, which is equal to 395.9

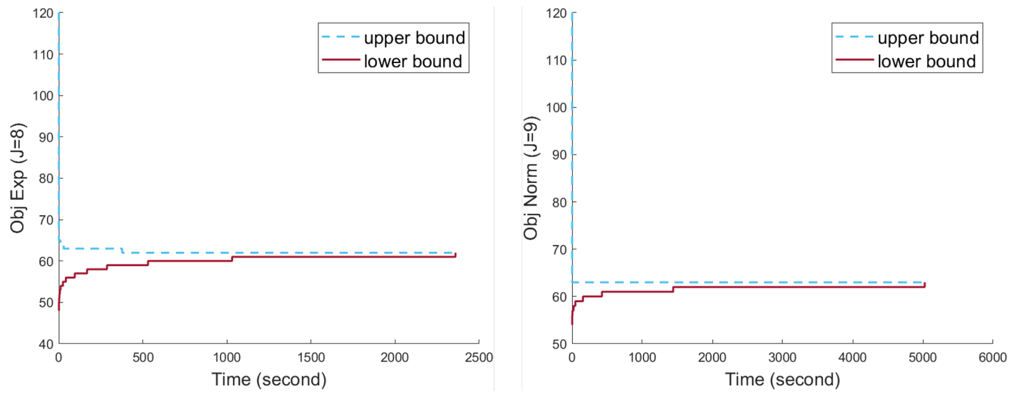

Figure 4: Upper bound and lower bound convergence. (8–stage Exp and 9–stage Norm experiment)

seconds on average, the proposed approach is superior in terms of the quality of the incumbent solution. It can also be seen that the performance of AHA-PS1 is not sensitive to the number of sampled solutions in each iteration. The comparison is also made for 5–stage, 10–stage, 15–stage Norm cases, each with 5 sample paths, using 10 as the number of sampled solutions in each iteration, AHA+PS1 is launched for five times for each sample path, and time limit is equal to 1800 seconds. The results, including the incumbent and time for visiting the incumbents, are shown in Table 2. Due to the noise, for different sample paths, and different launches of AHA+PS1, the incumbent could differ. The second and forth columns report all the incumbents. For example, in the 10–stages, the proposed algorithm always finds 71 as incumbent solution, while AHA-PS1 finds 71 in some cases and 72 in some others. In all the cases, the quality of incumbent of the proposed approach is not worse than AHA-PS1. As measure of efficiency, the time for visiting the final incumbent within the time limit is considered. Those times are reported in columns 3 and 5, where confidence interval with confidence level 95% is reported when the frequency of obtaining the incumbent is more than three in all the cases generated by the five sample paths, and the single values are shown directly when the frequency is only one or two. Comparing the contents in those columns, the proposed approach can provide the solution with same or better quality one order of magnitude faster than AHA-PS1. Moreover, the solutions provided by the proposed approach and time of visiting the incumbent with different sample paths are close to each other, which can also be seen in the 20–stage cases from Figure 5. This shows the stability of the output and efficiency.

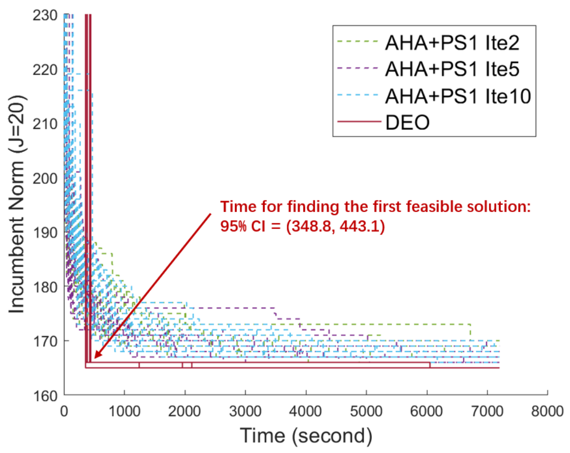

Figure 5: Comparison - incumbent convergence. (20–stage Norm experiment)

J	DEO		AHA-PS1 Ite10	
	Incumbent	Time for visiting incumbent (second)	Incumbent	Time for visiting incumbent (second)
5	32	2.2±0.2	32	61±10
10	71	20±2	71	990±246
			72	349±139
15	115	62, 87	116	1233
	116	82±17	117	996±264
			118	1071±268
			119	555
			121	1528
			122	1440

Table 2: Comparison of 5-stage, 10-stage, 15-stage Norm cases (5 instances for each case).

5 DISCUSSION AND CONCLUSION

This work proposes a sample–path average algorithm for solving the resource allocation problem with performance constraints in queueing systems. The algorithm is based on a MIP whose feasible region is defined by feasibility cuts, which are iteratively generated and added to the MIP. The main focus of this paper is how to generate and manage the feasibility cuts aiming at finding good solution at early time of the solution process and guaranteeing the global optimality at its termination. Gradient–based approximate cuts are first proposed, where the gradient can be estimated through simulating only one solution, which saves the simulation effort in finding good feasible solutions. Then, logic–based exact cuts are applied for finding the global optimum. Moreover, even though the gradient is approximate, it hints the potential bottleneck of the system, which enables to find tighter exact cuts. Numerical analysis has shown that the proposed approach can solve up to 9–stage problems within two hour time limit. A feasible solution with good quality, which is proved to be the global optimum at the end, can be found at very early time. Comparing with state-of-the-art approach, the proposed approach can provided better solution within shorter time. The superiority is significant especially in higher dimension problems. Furthermore, empirical study also shows that the solution quality and the efficiency of the proposed approach is not sensitive to the sampling noise.

Since the optimization is guided by mathematical models, simulation budget can be greatly saved. The main limitation of the proposed approach, from an efficiency point of view, is the fact that MIP has to be exactly solved at each iteration. However, the vast literature on mathematical programming can be used to improve such algorithm aspect.

Moreover, based on the feasible solutions found during the early phase of the solution process, local optimal solutions can be found by simple neighborhood enumeration. Thus, this algorithm can be also used as an efficient locally convergent SAA algorithm, even though it shares the same drawback of any SAA algorithm in the situation of stochastic constraints, i.e., the convergence is achieved only if the sample size is infinite. This drawback can be dealt with by retrospective approximation, consisting of a sequence of SAA for relaxed problems, where the sample size is smartly chosen (Pasupathy 2010; Nagaraj and Pasupathy 2014).

Even though it seems that the proposed approach can be used in various systems, since its derivation and the algorithm do not rely on strict assumptions, the boundary of its application is still vague. Specifically, which types of systems/networks and what performance indicators can be handled? Future effort will be dedicated to finding the prerequisite that a system and performance indicators must have to be dealt with as proposed in this work. Moreover, more numerical studies on different cases, such as call center staffing, and sensitivity analysis on the parameter α are needed to improve the study on the proposed approach. All this will be the subject of future research.

REFERENCES

Buzacott, J. A., and J. G. Shanthikumar. 1993. *Stochastic Models of Manufacturing Systems*, Volume 4. New Jersey: Prentice Hall Englewood Cliffs.

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

Geoffrion, A. M. 1972. "Generalized Benders Decomposition". *Journal of Optimization Theory and Applications* 10(4):237–260.

Hong, L. J., and B. L. Nelson. 2006. "Discrete Optimization via Simulation Using COMPASS". *Operations Research* 54(1):115–129.

Nagaraj, K., and R. Pasupathy. 2014. "Stochastically Constrained Simulation Optimization on Integer-Ordered Spaces: The cgR-SPLINE Algorithm". *Submitted to Operations Research*.

Norkin, V. I., G. C. Pflug, and A. Ruszczyński. 1998. "A Branch and Bound Method for Stochastic Global Optimization". *Mathematical Programming* 83(1-3):425–450.

Park, C., and S.-H. Kim. 2015. "Penalty Function with Memory for Discrete Optimization via Simulation with Stochastic Constraints". *Operations Research* 63(5):1195–1212.

Pasupathy, R. 2010. "On Choosing Parameters in Retrospective-Approximation Algorithms for Stochastic Root Finding and Simulation Optimization". *Operations Research* 58(4-part-1):889–901.

Shi, L., and S. Ólafsson. 2000. "Nested Partitions Method for Global Optimization". *Operations Research* 48(3):390–407.

Wang, H., R. Pasupathy, and B. W. Schmeiser. 2013. "Integer-Ordered Simulation Optimization Using R-SPLINE: Retrospective Search with Piecewise-Linear Interpolation and Neighborhood Enumeration". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 23(3):1–24.

Xu, J., B. L. Nelson, and L. J. Hong. 2013. "An Adaptive Hyperbox Algorithm for High-Dimensional Discrete Optimization via Simulation Problems". *INFORMS Journal on Computing* 25(1):133–146.

AUTHOR BIOGRAPHIES

MENGYI ZHANG is PhD candidate of Department of Mechanical Engineering at Politecnico di Milano, Italy. Her research interests include simulation optimization of manufacturing systems. Her email address is mengyi.zhang@polimi.it.

ANDREA MATTA is Professor at Politecnico di Milano, where he currently teaches integrated manufacturing systems and manufacturing. His research area includes analysis and design of manufacturing and health care systems. He is Editor-in-Chief of Flexible Services and Manufacturing Journal. His email address is andrea.matta@polimi.it.

ARIANNA ALFIERI is Professor at Politecnico di Torino, where she currently teaches production planning and control and system simulation. Her research area includes scheduling, supply chain management and system simulation optimization. Her email address is arianna.alfieri@polito.it.