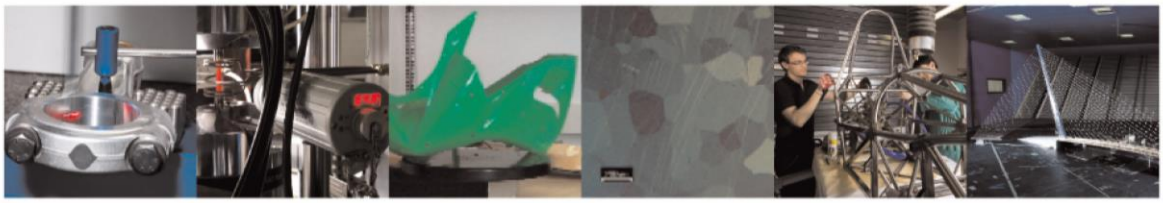


The time buffer approximated Buffer Allocation Problem: A row-column generation approach

Alfieri, A.; Matta, A.; Pastore, E.

DIPARTIMENTO DI MECCANICA ■ POLITECNICO DI MILANO
via G. La Masa, 1 ■ 20156 Milano ■ EMAIL (PEC): pecmecc@cert.polimi.it
http://www.mecc.polimi.it
Rev. 0

Abstract

One of the main problems in production systems is the buffer sizing. Choosing the right buffer size, at each production stage, that allows to achieve some performance measure (usually throughput or waiting time) is known as *Buffer Allocation Problem* (BAP), and it has been widely studied in the literature. Due to its complexity, BAP is usually approached using decomposition methods, under very strict system assumptions, or using simulation-optimization techniques. In this paper, the approximated mathematical programming formulation of the BAP simulation-optimization based on the time buffer concept is used. Using this approximation, buffers are modeled as temporal lags (*time buffers*) and this allows to use Linear Programming (LP) instead of Mixed Integer Linear Programming (MILP) models. Although LP models are easier to solve than MILPs, the huge dimension and the complex solution space topology of the time buffer approximated BAP calls for *ad hoc* solution algorithms. To this purpose, a row-column generation algorithm is proposed, which exploits the theoretical properties of the time buffer approximation to reduce the solution time. The proposed algorithm has been compared with a standard LP solver (ILOG CPLEX) and with a state-of-the-art MILP solver and it proved to be better than the LP solver in most of the cases, and more robust than the MILP solver with respect to computation time. Moreover, the LP model (for flow lines) is able to solve the BAP also for assembly/disassembly lines.

Keywords: Buffer allocation, simulation-optimization, assembly lines, math programming, row and column generation, exact method

1 Introduction

1.1 Buffer Allocation Problem

In multi-stage production systems, intermediate buffers are key elements to achieve a good system performance, especially in terms of throughput and waiting time. The presence of buffers before and after each stage allows to avoid, to some extent, the propagation of blocking and/or starvation between stages. Both these conditions are typical of queuing systems in general, and can occur when, for example, a server in a stage is failed and cannot process entities. In this case, assuming the extreme situation of no intermediate buffer, the stages following the failed stage are not fed (and then become *starved*), while the ones preceding it cannot unload the entity they have just processed (and then become *blocked*). Blocking/starvation situations may also occur when processing times are highly variable. In this case, a stage might be still processing an entity while the others (i.e., the preceding and the following stages) have already completed their job. In this case, blocking and starvation propagate in the

system exactly as the case of failed server.

With blocking/starvation, the throughput rate decreases (i.e., less entities are completed per time unit), the waiting time in front of the entry stage increases (or the balking rate increases if entities are not willing to wait), and the production lead time inflates (Hopp & Spearman, 2011; S. Gershwin, 1994; C. Papadopoulos, O'Kelly, Vidalis, & Spinellis, 2009).

To reduce blocking/starvation occurrences, the sources of variability should be reduced (Hopp & Spearman, 2011; S. Gershwin, 1994) (however, this issue strictly depends on the type of system), adeguate maintenance policies can be developed (Nahas, 2017; Renna, 2019), or intermediate buffers can be added. With buffers, if a server is failed or busy with the processing of an entity, then the previous and the next stages can keep processing. Specifically, the previous stages can unload the processed entities and put them in the buffer after the stage, while the next stages can process the entities accumulated in the buffer before the stage. Intermediate buffers are not able to completely avoid blocking/starvation, however the probability of their occurrence can be reduced, and the larger the buffer, the smaller the probability. Since increasing the buffer dimension increases the costs of the space and of immobilized capital, a trade-off between buffer dimension and costs must be evaluated. Also, other indirect costs are related to the increase of lead times and to the poor responsiveness for quick changes. Thus, selecting the dimension of the intermediate buffers is critical in the design phase of manufacturing systems. This problem, known as the *Buffer Allocation Problem* (BAP), is largely studied in the production system research (e.g., Buzzacott & Shanthikumar, 1993; S. Gershwin, 1994; C. Papadopoulos et al., 2009; Demir, Tunali, & Eliiyi, 2014).

In this paper, the BAP is considered for assembly/disassembly systems in production plants. Assembly/disassembly systems are a particular case of multi-stage production systems with serial and parallel stages but without routing decision to take for the parallel section of the system. In an assembly system, there are multiple first stages. When an order for a part is received, all its components enter the first stages, pass through the other intermediate stages (that can be different for each components) exactly in the same order, where they are worked in parallel, and then are assembled at the assembly stage. Once components are joined to form the part, the part can immediately leave the system, or it can be worked in other stages (that can be considered a transfer line) and, eventually, leaves the system from the last stage. Disassembly system, on the contrary, can be seen as transfer line in the first section, i.e., all the parts enter the system from the unique first stage, and then pass through the other intermediate stages exactly in the same order till the disassembly stage is reached. At the disassembly stage, the part is disassembled in its components that can be worked in subsequent stages in parallel until they reach the final stages (one for each component), where components leave the system. Assembly/disassembly systems, as transfer lines, are common

in many industrial sectors such as automotive, electronic and mechanical component production.

1.2 Methods for the Buffer Allocation Problem

The allocation of the buffer space between machines in a transfer line is an NP-hard problem, both for reliable (A. Dolgui, Eremeev, & Sygaev, 2010) and unreliable lines (A. Dolgui, Eremeev, Kovalyov, & Sigaev, 2013; A. B. Dolgui, Eremeev, Kovalyov, & Sigaev, 2018). According to the classification in Weiss, Schwarz, and Stolletz (2018), it can be solved through explicit solutions (Basu, 1977; Martin, 1990; L. Li, Qian, Du, & Yang, 2016), integrated optimization methods (Alfieri & Matta, 2012; Stolletz & Weiss, 2013; Weiss & Stolletz, 2015; Kolb & Göttlich, 2015), and iterative optimization methods (C. Papadopoulos, O'Kelly, & Tsadiras, 2013; Kose & Kilincci, 2015; S. B. Gershwin & Schor, 2000; Harris & Powell, 1999; H. Papadopoulos & Vidalis, 1998; Spinellis & Papadopoulos, 1998). As transfer lines can be considered a special case of assembly/disassembly systems, the allocation of the buffer space between machines in such systems is an NP-hard problem too, hence it is also usually solved by similar techniques (e.g., Hemachandra & Eedupuganti, 2003; Powell & Pike, 1998; Yamada & Matsui, 2003; Altiparmak, Dengiz, & Bulgak, 2007).

Under strict assumptions on processing times, i.e., when systems (mainly transfer lines) are characterized by deterministic or exponentially distributed processing times, decomposition strategies can be used to evaluate system performance. A few, non exhaustive, examples can be found in S. B. Gershwin (1987); Tolio and Matta (1998); J. Li and Meerkov (2009). These approaches are more efficient since they can lead to closed form solutions or fast performance evaluation algorithms. However, real systems unlikely satisfy these assumptions. In the case of general processing time distribution and more complex system layouts, analytical methods are not accurate and iterative optimization methods are the most common approach to deal with the BAP. The iterative methods find the solution by decoupling the problem into a generative and an evaluative procedure. Among them, the simulation-optimization approach is usually exploited, because of the structural properties of the problem (Seo & Lee, 2011; Alfieri, Matta, & Pastore, 2016; Weiss & Stolletz, 2015).

Traditional simulation-optimization approaches consist of two separate modules: a *simulation* module and an *optimization* module. These two modules work in an iterative scheme, alternating optimization and simulation steps. Specifically, in the optimization step, the optimization module is used to select a buffer configuration that is given in input to the simulation module. In the simulation step, the simulation module evaluates the feasibility and the performance of the input buffer configuration and gives such information to the optimization module that re-optimizes including new information. The simulation output, however, is affected by noise because of random events occurring at the machine level. This procedure is repeated until the optimal solution is found or some

predefined stopping condition is met (Spall, 2003; Kleijnen, 2008).

In simulation-optimization, the simulation module is usually a discrete event simulator (a computer code implemented in a specific simulation languages), while various approaches can be used in the optimization module. Among the most known and used approaches to optimization in simulation-optimization schemes (Fu, 2002) there are: response surface methodology (Kleijnen, 1998; Myers, Montgomery, & Anderson-Cook, 2009; Motlagh, Azimi, Amiri, & Madraki, 2019), stochastic approximation (Fu & Hu, 1997; Ho & Cao, 1991), random search (Andradóttir, 1996; Zabinsky, 2003), sample path optimization (Rubinstein & Shapiro, 1993), ranking and selection/ordinal optimization (Banks, J.S. Carlson, & Nicol, 2000; Ho, Cassandras, Chen, & Dai, 2000), and mathematical programming (Fu, Glover, & April, 2005). All the cited optimization approaches do not consider the system dynamics and are unable to evaluate the system performance, thus they need a simulation module. Simulation is affected by noise because of random events occurring at machine level. Also, the simulation module is used as a black-box, i.e., a simple performance evaluator.

In the last decade, however, a new simulation approach different from the simulation of Discrete Event Systems (DESs) has been proposed (Schruben, 2000; Chan & Schruben, 2008). This alternative approach is based on mathematical programming. A mathematical program is used to represent the system dynamics by means of constraints and the optimal solution of the mathematical program represents the so-called *system trajectory*. In other words, the arrival and completion times of each entity in each stage of the system are the same as those found with the standard simulation. Hence, system performance and feasibility can be easily evaluated in a unique model.

As mathematical programming has been used both in the optimization ad in the simulation module, integrated models that contain both the optimization criteria, the system dynamics and the performance measure have been developed (Matta, 2008). Solving such integrated simulation-optimization models corresponds to the alternating between the optimization and simulation modules in the iterative scheme previously described. Hence, using the integrated simulation-optimization mathematical programming models, a single simulation-optimization iteration is necessary (Weiss & Stolletz, 2015; G., Alfieri, & Matta, 2015). However, the solution of the integrated mathematical programming model requires a high computational effort since the models are no longer Linear Programs (LP) (as it happens in pure simulation (Alfieri & Matta, 2013)): the optimization component usually introduces binary or integer variables and the models become Mixed Integer Linear Programming (MILP) models.

1.3 Contribution

An approximation to reduce MILP models into LP models for the simulation-optimization of multi-stage production systems has been proposed in Alfieri and Matta (2012). The approximation is based on the so-called *time buffers* concept, which represents queues (i.e., buffers) *in time* instead of *in space*. Time buffers are temporal lags between pairs of entities. A time buffer represents the delay or the anticipation with which an entity can enter a stage with respect to the time another entity exits the same stage. It implicitly represents the availability or the unavailability of free slots (buffer spaces) in the considered stage.

Since time buffers are time variables, they can be represented by continuous variables, differently from the case of space buffer, which needs integer variables for the selection of the best buffer capacity. From the solution of the time buffer approximated BAP, a feasible solution for the exact integer BAP can be obtained and it has been shown to be very close to the optimum (Matta, 2008; Alfieri & Matta, 2012). Unfortunately, also the LP approximated models are not easy to solve (although they are easier than MILPs) due to the huge number of variables and constraints, and to the topology of the graph underlying the BAP approximated model. For this reason, the standard simplex method cannot be used and *ad hoc* algorithms must be designed.

In Alfieri et al. (2016) a column generation algorithm has been proposed to solve the approximated BAP for transfer lines. Column generation is an exact solution method, based on the decomposition of the original problem in a *Restricted Master Problem* (RMP) and a *Pricing Problem* (PP), and effectively used in many contexts (Barnhart, E.L.Johnson, Nemhauser, Savelsberg, & Vance, 1998; Castano, Rossia, Sevaux, & Velasco, 2014; Lubbecke & Desrosiers, 2005; Wolsey, 1998; Venkateshan & Mathur, 2011). Starting from an initial subset of variables, the PP is used to check if other variables (i.e., columns) can be added to the RMP. The information needed by the PP for the column evaluation are collected from the solution of the RMP.

In this paper, the work of Alfieri et al. (2016) is carried on by (i) extending the mathematical model to the case of assembly/disassembly multi-stage production systems and (ii) developing a row-column generation algorithm to solve efficiently the BAP. While in the column generation approach proposed in Alfieri et al. (2016) the RMP includes a subset of variables of the problem but all the constraints, in the row-column generation here proposed the RMP includes a subset of variables and only the subset of constraints related to them. Hence, at each step, rows (i.e, constraints) are added while adding columns (i.e., variables). Starting with a reduced number of constraints and variables, and adding both of them only if needed, the problem dimension is smaller than the one solved by the column generation in Alfieri et al. (2016) and, hence, the efficiency of the algorithm is enhanced. This makes the row-column generation usable for systems as the assembly/disassembly ones, which are more

general (and complex) than the transfer lines.

The proposed algorithm has been tested on randomly generated instances. Results showed that the proposed algorithm almost always outperforms ILOG CPLEX, a commercial available LP solver. Moreover, it has been compared to the state-of-the-art MILP solver proposed by Weiss and Stolletz (2015) in terms of computation time and optimal buffer capacity. Results show that the proposed algorithm finds an over-estimated buffer capacity; however, in some cases, it solves the BAP problem ten times faster than the exact method of Weiss and Stolletz (2015).

As the accuracy of the approximate solution has already been investigated in the previous works (Matta, 2008; Alfieri & Matta, 2012), the aim of this research is to improve the solution of the time buffer approximated BAP from a computation time perspective and to extend it to more complex multi-stage systems (namely, assembly/disassembly systems) and not to compare the approximated solution to the optimal solution of the exact BAP.

The contribution of the paper is twofold. It extends the work on time buffer approximation for the BAP to assembly/disassembly systems. Second, it improves the column generation algorithm of Alfieri et al. (2016) by proposing a more efficient row-column generation algorithm. The proposed algorithm, as the one presented in Alfieri et al. (2016), exploits the properties of the time buffer variables (Alfieri & Matta, 2012) to enhance the computational efficiency of the pricing problem.

1.4 Outline

The remainder of the paper is structured as follows: the time buffer concept and the approximated model are introduced in section 2. Section 3 presents the proposed row-column generation algorithm. The application of the algorithm on randomly generated instances and the comparison of the ad-hoc algorithm with a standard LP solved (namely ILOG CPLEX) and with the state-of-the-art MILP solver by Weiss and Stolletz (2015) is discussed in section 4. Section 5 concludes the paper.

2 Formal models for the Buffer Allocation Problem (BAP)

Before describing the mathematical programming models for the BAP in assembly/disassembly systems, the key notations and definitions are introduced. Although the proposed approach can be applied to many queueing systems, in the following production systems will be specifically considered. Hence, the terms machine and job will be used instead of server and entity.

2.1 Notation

The system is composed by J single machine stages, denoted by the index $j = 1, \cdots, J$ and characterized by a finite capacity c_{j-1}. The buffer B_{hj} represents the buffer between machines M_h and M_j. Notice that in the case of transfer lines or disassembly systems, there is a single first stage, otherwise, in the case of assembly systems, there could be multiple first stages. The opposite applies for the last stage (one in the case of transfer lines and assembly systems and multiple for disassembly systems). For this reason, index j just refers to the name of the stage and not to its position in the system. The entering buffers (i.e., the buffers before the first stages) are indicated by B_{0j} and are assumed to be infinite. This assumption avoids part balking, i.e., each part/job arriving at the system will be processed. Instead, no buffer is considered downstream the last stages so that the last stages are never blocked. The capacity of a stage includes both the slots in the buffer before the machine and the processing slots on the machine. Since the machines are assumed to own a single lot, buffer B_{hj} has a capacity equal to $c_{j-1} - 1$. The machine policy is the Blocking Before Service (BBS) (Dallery & Gershwin, 1992). Figure 1 depicts an example of assembly system.

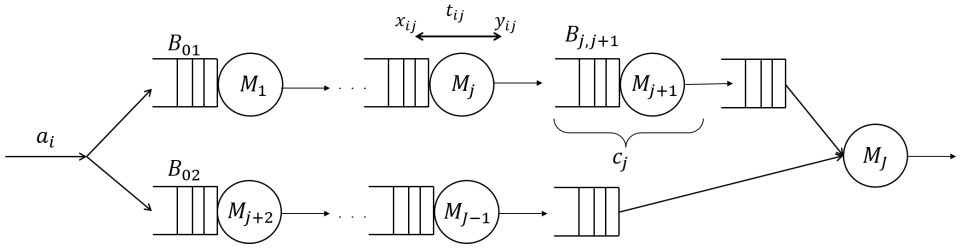


Figure 1: Assembly system with J stages.

Jobs entering the system are identical and denoted by index i, with $i = 1, \ldots, n$. Since jobs are identical and no sequencing decision is considered (i.e., jobs are served exactly in the same order they arrive to the line), the index i represents the i-th position in the sequence of jobs entering the system. Job i arrives in the system at time a_i (arrival time) and it is processed by all the machines exactly in the same order as the *machine configuration*. The machine configuration is given by the set $\mathscr{C}$, called *configuration set*, of the pairs of successive machines. Together with the configuration set $\mathscr{C}$, two more sets are needed to describe the system layout, namely, set $\mathscr{I}$ and set $\mathscr{F}$. The first contains all the initial machines, i.e., the entering stages of the jobs in the system; the second contains all the final machines, i.e., the machines after which the jobs leave the system. In the example of Figure 1, the configuration set is given by $\mathscr{C} = \{(M_1, M_2), \ldots, (M_j, M_{j+1}), (M_{j+2}, M_{j+3}), \ldots, (M_{J-2}, M_{J-1}), (M_{j+1}, M_J), (M_{J-1}, M_J)\}$, the set

of initial machines is $\mathscr{I} = \{M_1, M_{j+2}\}$, and the set of final machines is composed by $\mathscr{F} = \{M_J\}$.

The starting times, x_{ij}, and the finishing times, y_{ij}, together with the stage capacities, c_j, are the variables of the problem. In particular, variable x_{ij} denotes the time job i starts to be processed by machine j, while y_{ij} is the departure time of job i from stage j. According to the BBS rule, for each job, the starting and the departure times are linked by the following equation: $y_{ij} = x_{ij} + t_{ij}$, where t_{ij} is the parameter representing the processing time of job i at stage j. This relationship allows to use only one of the two variables. In particular, in this paper, variables y_{ij} will be used.

Arrival time a_i and processing time t_{ij} are known parameters for the mathematical programming models. They can be collected from a real system data or sampled from a probability distribution. No failure time is considered, i.e., machines are assumed to be perfectly reliable, or, identically, negligible repair times are assumed.

2.2 The exact simulation–optimization model

In the BAP, the objective is to find the best buffer capacity for each stage, i.e., the *buffer configuration* that allows to reach a predefined target value of some performance measure. In this paper, the throughput rate is considered as performance measure. This performance measure is related to the service level concept. In fact, the system service level can be related to the average completion time μ (the shorter this time, the higher the service level), i.e., in the case of a unique last stage J,

$$\mu = \frac{1}{n} \sum_{i=1}^{n} \frac{y_{iJ}}{i}, \tag{1}$$

and the average system throughput, θ, is the inverse of the average completion time, that is,

$$\theta = \frac{1}{\mu}. \tag{2}$$

Notice that the measure defined in equation (1) is slightly different from the one commonly used in the literaturek, i.e., $\mu = \frac{y_{nJ}}{n}$ (Alfieri & Matta, 2012). With the new formulation of equation (1), the average completion time is not strictly related to the last job (as in Alfieri and Matta (2012)), rather it is weighted through all the jobs. This gives a more realistic estimate of the inter-departure time in the case of highly variable stochastic processing time. Also, no warm-up period is considered in the paper, i.e., no jobs are excluded from the average completion time computation.

Using the notation introduced above, the simulation-optimization mathematical programming model for the BAP is as follows and can be simply derived from the one proposed in (Alfieri & Matta, 2012) for the transfer

lines by introducing sets $\mathscr{I}$, $\mathscr{F}$ and $\mathscr{C}$:

$$\min \quad \sum_{(j,h)\in\mathscr{C}} \left(e_{jh} \cdot \sum_{k=L_{jh}}^{U_{jh}} z_{jhk} \cdot k \right) \tag{3}$$

$$\text{s.t. :} \quad y_{ij} \geq a_i + t_{ij} \qquad \forall i, \forall j \in \mathscr{I} \tag{4}$$

$$y_{i+1,j} - y_{ij} \geq t_{i+1,j} \qquad \forall j, i = 1, \ldots, n-1 \tag{5}$$

$$y_{ih} - y_{ij} \geq t_{ih} \qquad \forall i, (j,h) \in \mathscr{C} \tag{6}$$

$$y_{i+k,j} - y_{ih} \geq t_{i+k,j} - (1 - z_{jhk}) \cdot M \quad i = 1, \ldots, (j,h) \in \mathscr{C},$$

$$L_{jh} \leq k \leq U_{jh} \tag{7}$$

$$\sum_{k=L_{jh}}^{U_{jh}} z_{jhk} = 1 \qquad \forall (j,h) \in \mathscr{C} \tag{8}$$

$$\widehat{\mu} \geq \frac{1}{n} \sum_{i=1}^{n} \frac{y_{ij}}{i} \qquad \forall j \in \mathscr{F} \tag{9}$$

$$\widehat{\mu} \leq \mu^* \tag{10}$$

$$y_{ij} \geq 0 \qquad \forall i, \forall j$$

$$z_{jhk} = \{0,1\} \qquad \forall (j,h) \in \mathscr{C}, \forall k.$$

The z_{jhk} binary variables are used to select the capacity of the buffer between machines j and h. Specifically, z_{jhk} is equal to 1 if the capacity of buffer B_{jh} is equal to k (with $k = L_{jh}, \cdots, U_{jh}$), i.e., if buffer B_{jh} is able to contains k jobs. Parameters L_{jh} and U_{jh} are the lower and the upper bound, respectively, for the capacity of buffer B_{jh}.

The solution of the BAP is the buffer configuration that minimizes the total buffer cost (3) meanwhile assuring a maximum average completion time equal to μ^*, that is, a minimum throughput $\theta^* = \frac{1}{\mu^*}$ (equation (10)). Constraints (9) are used in case of disassembly systems to get as maximum average completion time the largest among the various branches in which each part separates. In the case of single last stage (transfer lines and assembly systems), constraints (9) simply reduce to constraint (1). The performance constraint (10) has been stated in terms of service level rather than directly in terms of throughput to avoid non-linearity. The unit cost of the capacity associated to buffer B_{jh} is represented by parameter e_{jh}.

Constraints (4), (5) and (6) define the dynamics of the production system. In particular, each job (or components, in case of assembly systems) cannot complete its process and leave the initial stage (i.e., the ones in set $\mathscr{I}$) before being arrived to the system and spent there a time equal to its processing time on that machine (4). Each machine can process only one job at a time (5) and each job can be processed only by a machine at a time (6).

The capacity constraints are represented by equations (7), which prevent a job from leaving a stage if the downstream buffer is full. In fact, if $z_{jhk} = 1$, then all the constraints related to buffer B_{jh} with assigned capacity equal to k are activated; otherwise, the constraint is redundant thanks to the big–M in the right hand side. As

stated in constraints (8), only one capacity k can be chosen for each buffer.

This model is a simulation-optimization model since it contains the performance constraints (10), the optimization object function (3) and the system dynamics and capacity constraints (constraints (4)-(7)).

2.3 The Time Buffer approximated simulation-optimization model

The exact model presented in section 2.2 uses the concept of *space buffer*, that is, the available space between two consecutive stages in which jobs can wait before being processed. To model the capacity, the z_{jhk} binary variables have been used, leading to a MILP model.

In Alfieri and Matta (2012), an approximation of the exact model, based on the *Time Buffer* (TB) concept, has been proposed.

2.3.1 The Time Buffer concept

A time buffer can be considered as the time a job can start to be processed *in advance* on a machine before the next one becomes available. It considers the time aspect of buffers. In fact, if a space buffer is available between stage j and stage h, jobs can be completed in stage j and then wait in buffer B_{jh} that machine M_h becomes available. The *waiting* activity is exactly the time aspect of the buffer.

Let assume that two jobs, a and b, must be processed in the sequence $a \rightarrow b$ (Figure 2). A time buffer s_{jh}

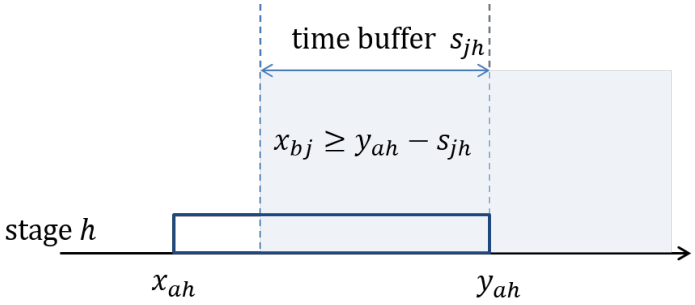


Figure 2: Time buffer between stage j and h

before server h and after server j constrains the starting time of job b at stage j, x_{bj}, and the departure time of job a from stage h, i.e., y_{ah}, in the following way:

$$x_{bj} \geq y_{ah} - s_{jh}. \tag{11}$$

According to equation (11), job a can start being processed at machine j only s_{jh} time units before job b leaves the stage h. In Figure 2, the gray area represents the possibility for a job to start its processing in a stage even if the machine in the next stage has not freed a space in its buffer because no item has finished there. In other words, it represents the fact that the next stage has not saturated its buffer capacity yet. The two extreme conditions of $s_{jh} = 0$ and $s_{jh} \to \infty$ model a perfect synchronization and a perfect decoupling of the two stages, respectively.

The TB concept can be considered a synchronization mechanism and can be adopted in every context where there is some queue in which objects have to wait.

Using the TB concept, an approximate simulation-optimization model for the BAP in transfer lines has been proposed in Alfieri and Matta (2012). Using the notation previously introduced, it can be extended to model assembly/disassembly systems:

$$\min \quad \sum_{(j,h)\in\mathscr{C}} \left(g_{jh} \cdot \sum_{k=1}^{K_{jh}} s_{jhk} \right) \tag{12}$$

$$\text{s.t.} \quad \text{(4)-(6), (9), (10)}$$

$$y_{i+k,j} - y_{ih} \geq t_{i+k,j} - s_{jhk} \qquad i = 1,\ldots,n-k, \quad (j,h) \in \mathscr{C}, k = 1,\ldots,K_j \tag{13}$$

$$y_{ij} \geq 0 \qquad \forall i, \forall j$$

$$s_{jhk} \geq 0 \qquad \forall (j,h) \in \mathscr{C}, k = 1,\ldots,K_{jh} \tag{14}$$

Differently from the exact optimization model, no more z_{jhk} binary variables are needed. The capacity of buffer B_{jh} is, in fact, implicitly represented by the s_{jhk} time buffers. In this case, k is the so-called *lag*, that is, the distance between two considered jobs, job i and job $i+k$. Each pair of jobs having the same lag k, could have its specific time buffer (e.g., the time buffer between job i and job $i+k$ could be different from the time buffer between job $i+1$ and job $i+1+k$). However, from a practical point of view, it is more useful to choose a time buffer for each pair of jobs having the same lag, that is, the time buffer is assumed to be the same for the pair i and $i+k$ and for the pair $i+1$ and $i+1+k$.

Parameters g_{jh} are the unit costs of the time buffers and the objective function (12) represents the minimization of the total time buffer cost. The system dynamics and the performance constraints, being based on the y_{ij} variables only, do not change with respect to the exact model. Instead, constraints (13) model the queue behavior (capacity constraints), substituting constraints (7) of the exact model: job $i+k$ has to wait starting its process at stage j until part i finishes its activity at stage h minus the time s_{jhk}.

The s_{jhk} variables represent times, hence they must be non negative as defined by equations (14). The K_{jh}

parameter is the maximum considered lag between stage j and h, i.e., jobs i and $i+K_{jh}+1$ are so distant one from the other that they have no relationship regardless of the capacity allocated to the buffer between stages j and h. Parameters K_{jh} are input parameters and their value depends on the considered system.

If, in the optimal solution, s^*_{jhk} is positive for some k, it means that a buffer B_{jh} of some capacity is needed. This consideration can be used to find a feasible solution for the exact BAP from the optimal solution of the approximated BAP.

Moreover, the time buffer variables have two interesting properties that will be exploited in the solution algorithm: 1) the throughput is a monotonic function of the time buffers; 2) the optimal time buffers are such that $s^*_{jhk} \geq s^*_{jh,k+1}$, for each (j,h) and k. The two properties have been proved in Alfieri and Matta (2012) and, for sake of conciseness, are not reported here.

The approximated simulation-optimization model just presented is an LP model. However, its dimensions are such that the standard simplex algorithm cannot be efficiently used for its solution.

To have an idea about the dimensions, consider a transfer line composed by J stages (i.e., $\mathscr{I}$ and $\mathscr{F}$ contains a single stage and $\mathscr{C}$ is composed by all the pairs of machines (M_j, M_{j+1})), and n jobs flowing through, and consider maximum lag $K_{j,j+1}$ for the pair of stages j and $j+1$ (i.e., the pair (j,h) is always equal to $(j, j+1)$). The dimension of the **y** and **s** variable arrays are:

- $dim(\mathbf{y}) = n \times J$;
- $dim(\mathbf{s}) = \sum_{j=1}^{J-1} K_{j,j+1}$,

while the number of constraints is

$$n + J(n-1) + n(J-1) + \sum_{j=1}^{J-1} \left(K_{j,j+1} \sum_{k=1}^{K_{j,j+1}} (n-k) \right) + 1.$$

As an example, assume the line has 10 stages, 5000 jobs are flowing through it and, at each stage, the maximum lag is equal to 40. In this case, the total number of variables is 25360 and the number of constraints is 71804791. The number of variables and constraints further increases when dealing with assembly/disassembly systems.

3 A row-column generation solution approach

To solve the approximated BAP, a *row-column generation* algorithm has been designed. Row-column generation is based on the principle that only a small fraction of the variables (the *columns*, namely $\mathscr{S}$) of a model will be in

the optimal solution and that there is a subset of constraints (the *rows*, namely $\mathscr{T}(\mathscr{S})$) depending on $\mathscr{S}$ that are needed to achieve the optimal solution, whereas the subset of constraints not depending on $\mathscr{S}$ can be somehow excluded. Starting from a subset of variables $\mathscr{S}_0$ and the related subset of constraints $\mathscr{T}(\mathscr{S}_0)$, the algorithm iteratively checks if other variables and constraints should be added to reach the optimal solution (Muter, Birbil, & Bülbül, 2013; Frangioni & Gendron, 2013; Sadykov & Vanderbeck, 2013; Maher, 2015). The general scheme of a row-column generation algorithm can be summarized as follows:

1. Start with a small set of variables $\mathscr{S}_0$ (corresponding to a *feasible solution*);
2. Find the set of constraints $\mathscr{T}(\mathscr{S}_0)$ (i.e., the set of constraints that are related to the variables in set $\mathscr{S}_0$);
3. Solve the LP with the restricted set of variables and constraints (*Restricted Master Program* (RMP));
4. Check if the solution is feasible for the complete LP, which includes all the constraints and all the variables, assigning value equal to zero to those not in the RMP (*optimality check*):
   - Yes $\rightarrow$ Optimal solution found (the algorithm stops);
   - No $\rightarrow$ Add new variables and the related constraints (*pricing problem*) and go to step 2.

3.1 The dual model for the approximated BAP

The primal model for the approximated BAP, presented in section 2, can be rewritten in a matrix form as follows:

$$\min \quad \mathbf{g}^T\mathbf{s} \tag{15}$$

$$\text{s.t.} \quad \mathbf{A}\mathbf{v} \geq \mathbf{b} \tag{16}$$

$$\widehat{\mu} \leq \mu^* \tag{17}$$

$$\mathbf{v} \geq 0 \tag{18}$$

Vector $\mathbf{v}$ is the set of variables ($\mathbf{y}$, $\mathbf{s}$), while $\mathbf{g}$ is the vector of the time buffer costs. The constraint matrix $\mathbf{A}$ is an $l \times m$ matrix, where l is the number of constraints, not including the throughput ones, and m is the number of decision variables. It can be represented as

$$\left[\begin{array}{c|c} \mathbf{A}_1 & \mathbf{0} \\ \hline \mathbf{A}_2 & \mathbf{A}_3 \end{array}\right]. \tag{19}$$

The two rows of matrix $\mathbf{A}$ refer to the system dynamics and maximum completion time definition constraints (4)-(6) and (9) and to the time buffer constraints (13), respectively. Instead, the two columns are related to the $\boldsymbol{y}$

and $\boldsymbol{s}$ variables, respectively. In particular, since the time buffers $\boldsymbol{s}$ are not involved in the system dynamics and completion time constraints, the second block of the first line is the null matrix. The m-dimensional vector $\mathbf{b} = \{b_1, b_2, \ldots, b_m\}$ contains all the parameters of the model, i.e., the arrival times a_i and the processing times t_{ij}.

The dual model in the matrix form is then the following:

$$\max \quad \mathbf{b}^T\mathbf{u} - \mu^* \cdot \rho \tag{20}$$

$$\text{s.t. :} \quad \mathbf{A}^T\mathbf{u} \leq \mathbf{g} \tag{21}$$

$$\mathbf{u} \geq 0, \quad \rho \geq 0 \tag{22}$$

The dual variables $\mathbf{u}$ are related to the primal constraints represented by the matrix $\mathbf{A}$, while ρ is the dual variable related to the throughput constraint (17). Specifying matrix $\mathbf{A}^T$ in its component blocks, constraints (21) takes the following form:

$$\left[\begin{array}{c|c} \mathbf{A}_1^T & \mathbf{A}_2^T \\ \hline \mathbf{0} & \mathbf{A}_3^T \end{array}\right] \mathbf{u} \leq \left[\begin{array}{c} \mathbf{0} \\ \\ \mathbf{g} \end{array}\right]. \tag{23}$$

$\mathbf{A}_3^T$ has dimension $r \times t$, where $r = \sum_{(j,h)\in\mathscr{C}} \left(K_{jh} \sum_{k=1}^{K_{jh}} (n-k)\right)$ and $t = \sum_{(j,h)\in\mathscr{C}} K_{jh}$ and every row in $\mathbf{A}_3^T$ is related to a particular s_{jhk} variable. This matrix will be the key element of the pricing procedure.

3.2 Row-column generation algorithm for the approximate BAP

As discussed in section 2.3, the approximated time buffer model contains two sets of variables: the $\boldsymbol{y}$ variables and the $\boldsymbol{s}$ variables. The y_{ij} variable represents the finishing time of job i at stage j. This implies that all the y_{ij} variables always take positive values and then all of them will enter the optimal solution. As for the $\boldsymbol{s}$, the properties proved in Alfieri and Matta (2012) and mentioned in section 2.3 assure that, at optimality, the s_{jhk} variables have values that are decreasing as k increases. This means that there will be a particular index $\widehat{k}$ such that $s_{jhk} = 0 \quad \forall k \geq \widehat{k}, \ \forall (j,h) \in \mathscr{C}$. From the above consideration, it can be stated that all the $\boldsymbol{y}$ variables will be included in the *initial solution* and the generation of columns will deal only with the generation of the $\boldsymbol{s}$ variables.

As in the LP, the $\boldsymbol{s}$ variables are included only in constraints (13), there will be a subset of them that will be excluded from the model, i.e., the ones containing the $\boldsymbol{s}$ variables not included in the RMP. In other words, the constraints in the RMP will be all the equations (4)-(6), (9), (10) and a subset of equations (13) .

The solution algorithm is described in the following. First, the mathematical model characterizing the Restricted Master Problem is presented. Then, the pricing problem, the optimality check procedure and the algorithm to find the initial solution are devised.

3.2.1 Restricted Master Program (RMP)

Let $\mathscr{S}_0$ be the set of the time buffer variables generated so far and let $\mathscr{T}(\mathscr{S}_0)$ be the set of the time buffer constraints related to $\mathscr{S}_0$. The RMP can be written as:

$$
\begin{aligned}
\min \quad & \textstyle\sum_{(j,h)\in\mathscr{C}} \left(\sum_{s_{jhk}\in\mathscr{S}_0} g_{jh} s_{jhk} \right) \\
\text{s.t.} \quad & (4)-(6) \\
& y_{i+k,j} - y_{i,h} \geq t_{i+k,j} - s_{jhk} \qquad i = 1,\dots,n-k,\ (j,h)\in\mathscr{C} \\
& \qquad\qquad\qquad\qquad\qquad\qquad k = 1,\dots,K_{jh}, \quad \forall s_{jhk}\in\mathscr{S}_0 \qquad (24) \\
& \widehat{\mu} \geq \tfrac{1}{n}\textstyle\sum_{i=1}^{n} \frac{y_{ij}}{i} \qquad \forall j\in\mathscr{F} \\
& \widehat{\mu} \leq \mu^* \\
& y_{ij} \geq 0 \qquad \forall i,\forall j \qquad (25) \\
& s_{jhk} \geq 0 \qquad \forall s_{jhk}\in\mathscr{S}_0 \qquad (26)
\end{aligned}
$$

The RMP is exactly as the complete approximated model with the only difference that the $\boldsymbol{s}$ variables are restricted to the subset $\mathscr{S}_0$ and the constraints in equation (24) are limited to the subset $\mathscr{T}(\mathscr{S}_0)$. At the beginning, $\mathscr{S}_0$ contains the s_{jhk} corresponding to the initial solution and $\mathscr{T}(\mathscr{S}_0)$ the time buffer constraints that include only the s_{jhk} included in $\mathscr{S}_0$. The RMP is solved and, if the optimality check fails, the pricing problem is used to add to $\mathscr{S}_0$ the variables (and the related time buffer constraints) that are needed to improve the solution. Then, the RMP is solved again, the optimality checked again until the check is passed (i.e., the original complete problem is solved at optimality).

3.2.2 Pricing problem

The pricing problem is used to find the variables (and the related constraints) to add and it is usually based on the dual information. In particular, in the case of BAP, the feasibility of the dual variables should be checked, i.e., it should be checked if the optimal dual variables of the RMP satisfy constraints (23). As the search is limited to the time buffer variables, only the dual constraints related to the $\mathbf{A}_3^T$ matrix must be checked.

Specifically, for each time buffer s_{jhk}, the dual constraint to be checked is:

$$\sum_{i=1}^{n-k} u_{ijhk} \leq g_{jh},$$

where u_{ijhk} is the dual variable related to the primal constraint $y_{i+k,j} - y_{ih} \geq t_{i+k,j} - s_{jhk}$. When, for a given triplet (j,h,k), the sum is bigger than g_{jh}, then the s_{jhk} variable must be added to $\mathscr{S}_0$.

Exploiting the property that assures, for the time buffers, non-increasing values in k for each $(j,h) \in \mathscr{C}$, the time buffers are added proceeding from the lowest values of index k to the highest ones. For instance, assume that, in the current iteration, the RMP has $\mathscr{S}_0 = \{s_{jhk}, \quad k = 1,\ldots,\widehat{k}, \quad (j,h) \in \mathscr{C}\}$ (where $\widehat{k} < K_{jh}$). Among all the dual constraints, almost surely only the ones with $k = \{\widehat{k}+1, \widehat{k}+2\}$ will be violated, thus implying that only variables s_{jhk} with $k = \{\widehat{k}+1, \widehat{k}+2\}$ will be able to enter $\mathscr{S}_0$ in the next iteration, i.e.,

$$\mathscr{S}_0 \leftarrow \mathscr{S}_0 + \{s_{jhk} \mid k = \widehat{k}+1, \widehat{k}+2; \quad \forall (j,h) \in \mathscr{C}\}.$$

This property empirically holds but it has not been theoretically proved. However, it can be exploited to speed up the pricing procedure without the risk of loosing optimality. In fact, if the optimality check failed in the last iteration, in the next iteration only time buffer variables s_{jhk} with $k = \{\widehat{k}+1, \widehat{k}+2\}$ will be added to $\mathscr{S}_0$.

3.2.3 Searching for an initial feasible solution

An initial solution can be found by exploiting the fact that the considered model is a simulation-optimization model. Due to the simulation component, the optimal buffer configuration should be a steady-state solution, which is independent from the number n of jobs. This means that the solution corresponding to a subset of jobs $n_0 << n$ can be considered as an approximation of the optimal solution.

Starting from this observation, the initial solution can be found by solving the complete primal model instanced with only $n_0 << n$ jobs. The solution can be infeasible because of the violation of the completion time constraints, which can be due, in turn, to a real infeasible problem (the requested throughput is too high for the system capacity in terms of processing times and hence no BAP can be solved on that system) or to a small buffer capacity, i.e., a too small value for some K_{jh} (the maximum lag). In this case, the complete model can be solved again after having increases the K_{jh}.

Once the model with n_0 flowing jobs is solved, the index

$$k_0 = \max_k \{k \mid \exists s_{jhk} > 0, \quad (j,h) \in \mathscr{C}\} \tag{27}$$

is stored and the RMP is instanced with $\mathscr{S}_0$ containing the following elements:

$$\mathscr{S}_0 = \{s_{jhk} \mid k \leq k_0, \quad \forall (j,h) \in \mathscr{C}\} \tag{28}$$

and $\mathscr{T}(\mathscr{S}_0)$ containing the following constraints:

$$y_{i+k,j} - y_{i,h} \geq t_{i+k,j} - s_{jhk} \quad i = 1,\ldots,n-k, \ (j,h) \in \mathscr{C}, k = 1,\ldots,K_{jh}, \quad \forall s_{jhk} \in \mathscr{S}_0. \tag{29}$$

Notice that the initial solution can be infeasible for the complete problem with all the n jobs as the buffer capacity (and hence the time buffers) depends on the variability of the arrival and processing times, which cannot be correctly estimated if the considered sample of jobs is too small. However, also in this case, feasibility can be reached by increasing the value of K_{jh}.

3.2.4 Optimality check - stopping condition

In each iteration, after the RMP is solved, the optimality check procedure controls if the solution found so far is optimal for the complete primal problem.

The optimality check consists in a feasibility control related to the time buffer constraints not included in the RMP. It works as follows. The solution found in the last iteration has been only considering the subset $\mathscr{S}_0$ of $\boldsymbol{s}$ variables and the related constraints $\mathscr{T}(\mathscr{S}_0)$. This means that there exists a set of constraints that have been left out from the problem (namely, $\mathscr{T}^L(\mathscr{S}_0)$). This set corresponds to the following equations:

$$y_{i+k,j} - y_{i,h} \geq t_{i+k,j} - s_{jhk} \quad i = 1,\ldots,n-k,\ (j,h) \in \mathscr{C}, k = 1,\ldots,K_{jh} \quad \forall s_{jhk} \notin \mathscr{S}_0. \tag{30}$$

In reality, for the $s_{jhk} \notin \mathscr{S}_0$, the model should contain the following equations,

$$y_{i+k,j} - y_{i,h} \geq t_{i+k,j} \quad i = 1,\ldots,n-k,\ (j,h) \in \mathscr{C}, \forall k \text{ such that. } s_{jhk} \notin \mathscr{S}_0. \tag{31}$$

Equations (31) rises from equation (30) forcing all $s_{jhk} \notin \mathscr{S}_0$ to be set to zero.

As the solution found so far is optimal for the problem that does not include these equations, the optimality check must control for the feasibility of this solution in the complete problem, i.e., if equations (31) are verified.

If all the equations in (31) are verified, then the actual solution is feasible and optimal also for the complete problem, hence the optimality is reached and the algorithm stops.

Algorithm 1 summarizes the complete column generation procedure.

4 Computational testing

Numerical tests have been performed to evaluate the efficiency of the proposed algorithm. A first experiment tests the computation time of the row-column generation algorithm discussed in section 3 and compares it with ILOG CPLEX, a commercially available LP software. The focus of the experiment is on computation times, since the proposed row-column generation is an exact algorithm, and, hence, it reaches, exactly as CPLEX, the optimum. A second experiment compares the proposed algorithm with the one proposed in Weiss and Stolletz

Algorithm 1 Column Generation algorithm

Initialisation.

Set n, J, n_0;

Use standard LP solver to solve the approximate optimization model with n_0 jobs and J stages;

Compute k_0 as in (27).

Column Generation.

Define $\mathscr{S}_0$ as stated in equation (28) and related $\mathscr{T}(\mathscr{S}_0)$;

Check= 0;

while Check= 0 **do**

Solve the RMP with $\mathscr{S}_0$ and $\mathscr{T}(\mathscr{S}_0)$;

if Optimality check is positive **then**

optimal solution found;

Check= 1;

else

Generate columns;

Update $\mathscr{S}_0$ with the new variables and $\mathscr{T}(\mathscr{S}_0)$ with the new constraints;

end if

end while

(2015) on real flow lines described in Tempelmeier (2003). The focus of this second experiment is comparing the LP approximation with the MILP exact model in terms of total buffer capacity and computation time. In the comparison, the LP is solved by the proposed row-column generation, whereas the MILP is solved with the state-of-the-art algorithm proposed in Weiss and Stolletz (2015), which exploits the Benders decomposition for MILP models with combinatorial cuts.

4.1 Approximated BAP experiment

The first experiment deals with the investigation of the computation time of the proposed algorithm. Various production systems, with different characteristics, have been tested. The computation times of the row-column generation and of CPLEX have been collected and compared, to assess the efficiency of the proposed algorithm. Before presenting the numerical results, the experimental settings are described in details.

4.1.1 Experiment settings

In the experiments, production systems with various characteristics have been considered. As assumed in section 2, the buffer(s) in front of the initial stage(s) has infinite capacity and, after been processed by the last machine(s), jobs immediately leave the system without restrictions or delays. The maximum lag K_{jh} for each pair of stages $(j,h) \in \mathscr{C}$ is assumed equal to 40. Moreover, for simplicity, no warm-up period has been considered, as previously defined (equation (1)).

Without loss of generality, it has been assumed that the unit cost of the time buffers is equal to 1 for each stages, i.e., $g_{jh} = 1 \ \forall (j,h) \in \mathscr{C}$, and that all the jobs are available at time 0, i.e., $a_i = 0 \ \forall i$. The first assumption corresponds to minimizing the total time buffer allocated to the system, instead of its cost, which is the same if the buffer capacity has the same cost in each stage, as it usually happens in real systems. The second assumption is related to the control of the variability in the system, which, in the case of $a_i = 0$, only depends on the variability of the processing times. In each experiment, the number of jobs considered to find the initial solution has been set to $n_0 = 1000$. The above discussed parameter values are summarized in Table 1.

Parameters		
Maximum lag $(\forall j,h)$	K_{jh}	40
Number of jobs for initial solution	n_0	1000
Unit cost of time-buffers	g_{jh}	1

Table 1: Parameter setting

The elements that influence the performance of the algorithm, beside the variability, are the dimensions of the problem, the saturation of the line, the configuration of the system and the existence of a bottleneck. Hence, these elements have been considered as the experimental factors and different levels have been tested. Table 2 reports the values considered for each experimental factor.

The first two factors, i.e., the number of stages J and of jobs n, are related to the dimension of the problem. The larger these values, the larger the number of variables and constraints, and then the more complex the problem. The dimension of the problem is also related to the maximum lag K_{jh} at each stage (which has been fixed for each experiment, as reported in Table 1).

The system configuration C deals with the configuration of the production system. Three configurations have been taken into account (assembly, disassembly and transfer line). The line configuration is a sequence of either

Factors		
Number of stages	J	$\{5, 10\}$
Number of jobs	n	$\{5000, 10000, 20000\}$
System configuration	C	{Assembly, Disassembly, Line}
Type of system	L	{bal, unb}
Saturation	R	$\{70\%, 90\%\}$
cv for t_{ij} distribution	c_v	$\{0.5, 1\}$

Table 2: Design of experiment

5 or 10 stations, with a linear flow. Figure 3 shows the assembly and disassembly configurations. In each graph of the figure, the circles represent the stages of the production system and the arrows indicate the flow of the jobs. When $J = 5$, in the assembly system (the graph on the top left of the figure) jobs enter the system through stages 1 and 3, move to stages 2 and 4 and finally got assembled in stage 5, then leave the system. Following the assumptions listed before, the buffers before stages 1 and 3 have infinite capacity and jobs leave the system as soon as they finish to be processed in stage $j = 5$. For the systems with $J = 10$, station $j = 9$ is the assembly stage, buffers before stages 1 and 5 have infinity capacity, and jobs leave the system after stage 10. The same holds for the disassembly systems, and their configurations are shown in the figure.

The type of system L represents, instead, the difference of processing times among the machines of the line. The value *bal* refers to the case of a balanced system (all the J stages have the same average processing time), while *unb* refers to the case of a bottleneck positioned at the beginning ($j = 1$) of the system. In the balanced case, an average processing time equal to 2 units of time has been considered for each machine, while, in the unbalanced case, the bottleneck has an average processing time equal to 4 units of time and the other machines equal to 2 units of time (in assembly systems, $j = 1$ is the bottleneck). In all the experiments and for all the machines, a LogNormal distribution has been assumed for the processing times.

The saturation R is the utilization rate of the bottleneck and controls the target throughput: the higher the saturation, the higher the achievable throughput, and the lower the maximum service level.

Finally, the coefficient of variation of processing times, c_v, is used to control the variability in the system. It is a statistical measure of the dispersion computed as the ratio between the standard deviation σ and the mean μ, i.e., $c_v = \frac{\sigma}{\mu}$. Given the average processing time μ and the coefficient of variation c_v, the standard deviation can be computed for both the bottleneck and the non-bottleneck machines.

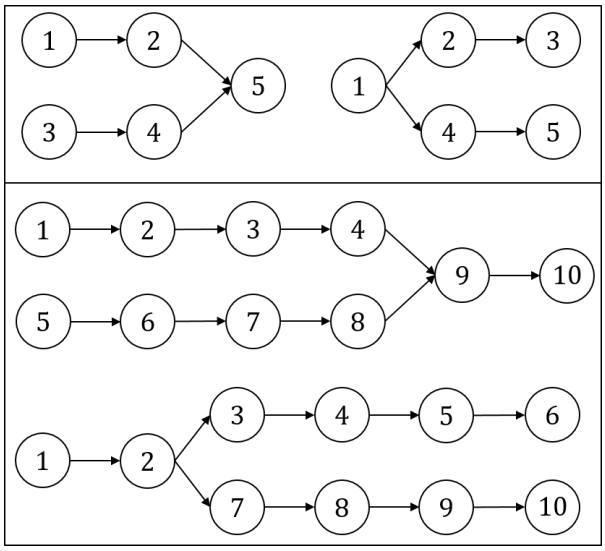


Figure 3: Assembly and disassembly configurations with $J = 5$ and $J = 10$.

The six factors lead to 144 combinations. For each combination, 30 instances have been generated, leading to 4320 experiments in total.

4.1.2 Numerical results

Each of the 4320 experiments has been solved using the designed algorithm and the ILOG CPLEX software. More precisely, for ILOG CPLEX, the *Barrier Algorithm* (an interior point algorithm) has been used, as (in preliminary tests) it has shown to be the fastest among the solution algorithms available in the ILOG CPLEX environment for the specific problem to be solved.

The row-column generation algorithm has been implemented with an object oriented paradigm in Java language on Eclipse platform. ILOG CPLEX 12.6 has been used both as solver for the complete problem (to compare with the proposed algorithm) and as solver for the RMP during the row-column generation iterations. The experiments have been carried out on a Core i7-7700K with 4.2 GHz and 32 GB RAM.

As previously stated, since both the algorithms (i.e, row-column generation and CPLEX barrier) are exact algorithms, the achieved solution is the optimal solution for both of them. Thus, in the following, the solution values are not reported and the algorithms are compared only in terms of computational efficiency.

Table 3 shows the average computation times (in seconds) and their variability (standard deviation), for

CPLEX and for the row-column generation, for different values of n. Each row of the table shows the average computation times for a specific value of J and n. The first three columns report the results for the three system configurations C, thus each computation time value is the average over all types of system L, all c_v, and all levels of saturation R with fixed n, J and C. The last column α shows the percentage of instances in which the row-column generation algorithm is more efficient than CPLEX. This percentage is always larger than 95% with $J = 5$ and larger than 87% with $J = 10$, which means that the proposed algorithm almost always outperforms the commercial solver.

J	**n**	**Assembly**		**Disassembly**		**Line**		**α**
		CPLEX	Row-Column	CPLEX	Row-Column	CPLEX	Row-Column	
5	5000	32.82 (6.14)	9.85 (6.13)	31.54 (9.39)	9.35 (5.77)	31.43 (5.66)	9.93 (6.31)	95.69 %
5	10000	81.12 (17.12)	20.73 (13.52)	87.69 (45.24)	19.49 (12.54)	80.89 (46.11)	19.43 (12.35)	99.17 %
5	20000	208.11 (88.72)	53.65 (32.65)	238.37 (175.69)	55.28 (30.04)	220.01 (243.12)	51.66 (28.23)	99.72 %
10	5000	85.34 (13.13)	36.60 (41.06)	84.76 (10.15)	36.43 (5.96)	89.18 (17.97)	47.11 (69.59)	87.50 %
10	10000	284.16 (532.79)	70.22 (70.78)	213.56 (916.73)	69.33 (70.92)	236.86 (108.97)	72.73 (80.77)	87.64 %
10	20000	792.20 (1024.10)	176.10 (154.68)	574.90 (328.12)	170.65 (146.80)	848.05 (1154.31)	163.19 (150.07)	87.92 %

Table 3: Computation times: average values $[s]$ (standard deviation) and percentage of efficient instances.

As expected, for both the algorithms, the larger the system and the number of jobs in it, the larger the computation time needed for finding the optimal time buffer configuration. However, the computation time of the row-column generation algorithm grows with a less steep slope with respect to CPLEX (Figure 4). This is an

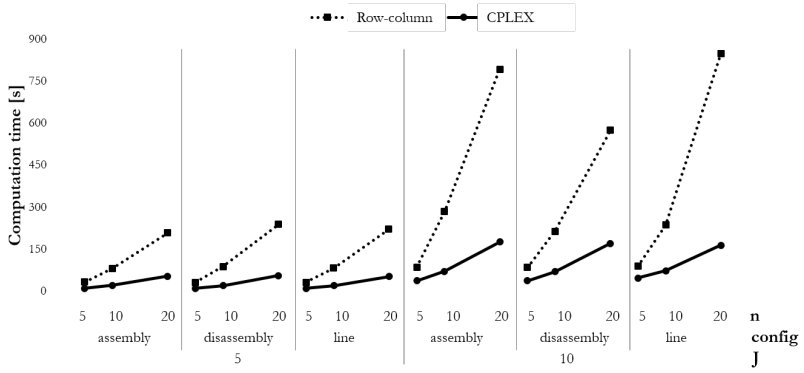


Figure 4: Average computation times: slopes.

important result since it implies that the difference in efficiency increases as n and J increase, i.e, the larger

the instance (e.g., the bigger the system, or the larger the number of jobs or the maximum lag), the larger the benefit in using the row-column generation algorithm. Moreover, the difference between the average CPLEX and row-column generation computation times is significant for each combination of factors. Also, for the line system configuration, the row-column generation algorithm improves the column-generation algorithm presented in Alfieri et al. (2016) by decreasing the computations time by 30-40 %.

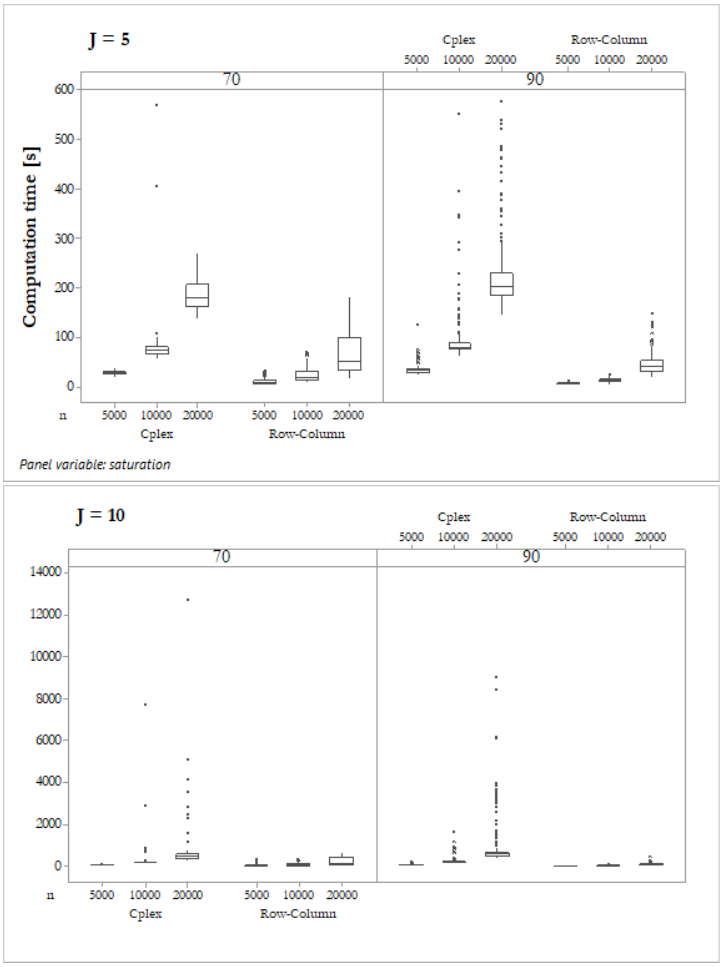


Figure 5: Analysis of the distribution of computational times.

From Table 3, the variabilities of the CPLEX and the row-column are quite different. Figure 5 shows the distribution of the computation times grouped by different J, saturation levels and numbers of jobs. Each graph is related to a value of J; for each graph, the two panels are related to the 70% and the 90% saturation, whereas

each boxplot refers to a specific n value and a specific algorithm. The high variability related to the CPLEX times is explained by the various outliers (indicated by * in the graphs). In various cases, CPLEX takes a large amount of time to solve the mathematical problem, especially for large number of stages and of jobs and a high saturation percentage. This is coherent with the mathematical model, as the number of jobs and stages influence the number of variables of the LP, whereas a large saturation let the throughput constraint (equation (9)) be tighter.

Further analyses have been made on the percentage of instances in which the row-column generation algorithm has been more efficient than CPLEX. Specifically, Table 3 shows that for a few instances, the proposed algorithm is outperformed by CPLEX. These instances are all characterized by a balanced system and high coefficient of variation of processing times; moreover, high saturation makes the row-column generation worse than CPLEX sometimes. Figure 6 shows the comparison between CPLEX and the row-column algorithm computation times: the assembly system characterized by balanced stages and high coefficient of variation is considered as an example; the results are shown separately for the two saturation levels. Each point of the graphs displays the result of a single instance in terms of CPLEX and row-column algorithm computation times: if the point is below the straight line, the row-column algorithm time is smaller than that of CPLEX, and vice-versa. The graphs show that all the instances with low saturation have row-column times largely smaller that CPLEX, while for the cases of high saturation a large portion of points are above the line, meaning that the row-column algorithm time resulted to be larger than CPLEX.

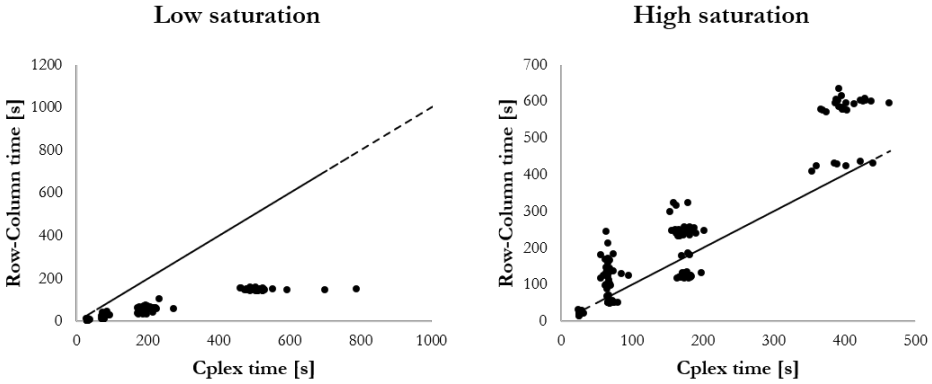


Figure 6: CPLEX and row-column algorithm computation times: disassembly balanced system, $c_v = 1$.

An ANOVA test has been run both for CPLEX and the proposed algorithm computation times, to understand which factors influence the computation effort to solve the problem.

The results show that, for a significance level of 99.9 %, both CPLEX and the row-column algorithm computation time are influenced by the system type, the number of jobs and stages, and the saturation, whereas only the row-column algorithm computation time is influenced by the variability of processing time. The common influence of the number of jobs flowing in the system, the number of stages and the saturation was expected. The larger the number of jobs or the number of stages, the larger the number of variables and constraints characterizing the mathematical model, indeed. Also, about saturation, higher saturation leads to a tighter throughput constraint, thus the solution is harder to be found. Instead, the coefficient of variation of the processing times seems to influence the row-column generation computation time, but not the CPLEX time. The system type (i.e., balanced or unbalanced system) influences both the algorithms, whereas the system configuration seems to have no influence.

4.2 Exact and approximated BAP in real lines

In the second experiment, the proposed row-column generation for the approximated BAP is compared with the state-of-the-art Benders decomposition proposed in Weiss and Stolletz (2015) to solve the exact BAP (namely, SOA). Only flow lines are considered in this experiment, as the SOA is not able to solve the BAP for assembly/disassembly systems. Moreover, real cases reported in Tempelmeier (2003) are taken into account to evaluate the efficiency of the two solution methods.

In the following, the experimental settings are discussed and the results of the experiment are then presented.

4.2.1 Experimental settings

In the experiments, the realistic flow lines presented in Tempelmeier (2003) are considered. Specifically, lines from A to D are tested. In Tempelmeier (2003), each line is associated to deterministic processing times and failures. In this paper, stochastic processing times were included, by considering LogNormal distributions (as in the previous experiment). As failures are not included in the formulation of the BAP model used in the paper, the deterministic processing times of Tempelmeier (2003) have been divided by the availability of the machines. Thus, the LogNormal distributions of machine processing times have mean equal to the deterministic values divided by the availability, and coefficient of variation varying according to the design of experiment discussed in the following. The machine mean processing times and the configuration of the four lines are shown in Figure 7. For each graph, the bars represent machines (for instance, line A has 19 bars and hence 19 machines), and the height of the bar is the average processing time (expressed in minutes). From the figure, line B is the longest,

with 23 machines, whereas line C is the shortest, with 8 machines. With 8 to 23 machines, exact methods hardly find the optimal solution in a short time (Weiss et al., 2018).

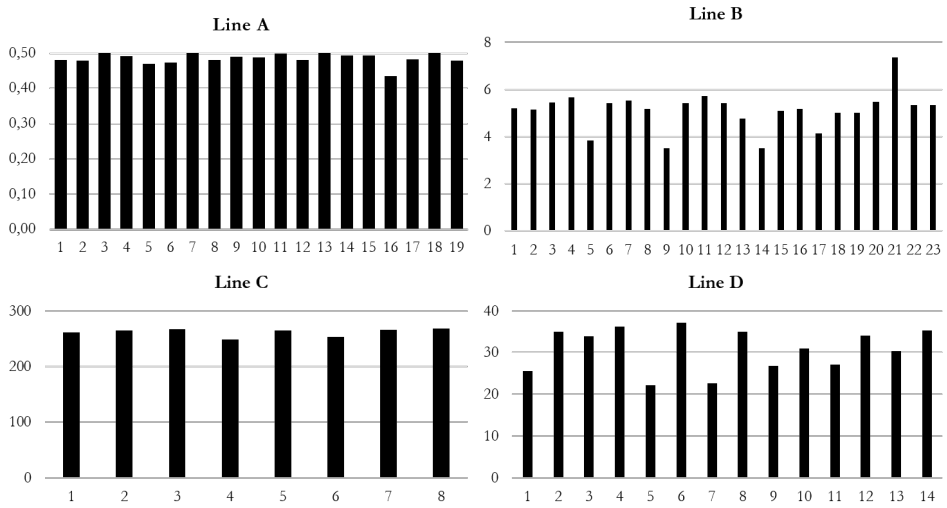


Figure 7: Characteristics of the simulated lines from Tempelmeier (2003): the horizontal axes represent the machines of the lines, and the vertical axes correspond to the average processing time of the machines.

In the experiment, the number of jobs has been fixed to $n = 250000$, to obtain a reliable buffer configuration. The target cycle time of each line has been chosen by simulating the line with the resulting stochastic processing times and with the buffer capacity given in Tempelmeier (2003). For the four lines, the total buffer capacity of each line of Tempelmeier (2003) is: 285 for line A, 459 for B, 131 for C, and 166 for D. As in the previous experiment, the unit cost of the time buffers for the row-column generation and that of the buffer capacity for the SOA are set to 1, and that all the jobs are available at time 0. Also, both the maximum lag K_{jh} of the row-column generation and the upper-bound of the buffer capacity at each stage for the SOA have been set to 200.

The factors of the experiment are: the line (i.e., line A, B, C, D of Figure 7), and the coefficient of variation of the processing times ($c_v = \{0.5, 1\}$, as in the first experiment). In total, 8 combination of factors are evaluated. The combinations and the associated target cycle times are shown in the left-part of Table 4. For each combination, 10 independent replicates have been run, for a total of 80 instances. For each instance, a budget of 18000 seconds (i.e., 5 hours) has been allowed to solve the problem (as long flow lines are hard to solve). The results are presented in the following section.

4.2.2 Numerical results

All the experiments have been solved by using the same computer used for the first experiment, and the SOA has been implemented in Java language in the same Eclipse platform of the row-column generation algorithm.

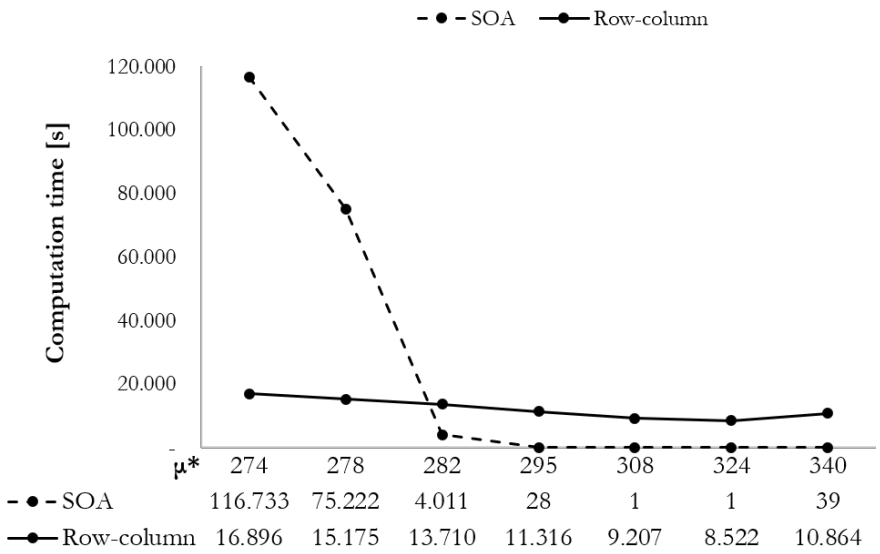
Line		c_v	Row-Column		SOA	
Line	μ^*		**comp. time**	β	**comp. time**	β
A	0.52	0.5	>18000	0%	>18000	0%
A	0.58	1	>18000	0%	>18000	0%
B	7.37	0.5	>18000	0%	>18000	0%
B	7.35	1	>18000	0%	>18000	0%
C	277.78	0.5	15599	90%	>18000	0%
C	303.03	1	14014	70%	>18000	0%
D	39.68	0.5	15477	40 %	>18000	0%
D	46.08	1	>18000	0%	>18000	0%

Table 4: Mean computation times for real lines.

The average computation times (over the replicates) for all the experimented systems are shown in Table 4. For each algorithm, the computation time and the percentage of replicates solved under the threshold of 18000 seconds (namely, β) are shown. While the SOA is never able to solve the MILP problems within the five hours, the row-column generation for the time buffer BAP is able to solve a small portion of the instances. This result reflects the complexity for exact algorithms to solve the BAP with realistic dimensions. Specifically, both the solvers are not able to solve any of the replicates of lines A and B; however the row-column generation can solve a portion of the instances of lines C and D. As line C and D are the shortest line, this result confirms that the computation time of the row-column generation largely depends on the dimension of the problem (i.e., from the number of stages and jobs, in this case). Instead, from the results in Weiss and Stolletz (2015), the computation time of the SOA seems to depend on the magnitude of the optimal total buffer capacity, which for all the systems was extremely large (as previously mentioned).

To confirm and deepen the comparison between the two algorithms, another experiment has been carried out. As the row-column generation seems to depend on the number of stages and the SOA on the total buffer capacity, line C is optimized with different target cycle times. In this way, the two algorithms are tested with various values

of optimal total buffer capacity.



μ*	274	278	282	295	308	324	340
SOA	116.733	75.222	4.011	28	1	1	39
Row-column	16.896	15.175	13.710	11.316	9.207	8.522	10.864

Figure 8: Computation times with varying μ^*.

The target cycle time μ^* has been varied between 273.97 and 340.45, and results about the computation time and the optimal buffer capacity have been collected for both the row-column generation and the SOA. Figure 8 shows the comparison of the performance of the two algorithms. In the figure, the horizontal axis indicates the target cycle time μ^*, and the vertical axis shows the computation time (expressed in seconds). The two rows below the horizontal axis show the buffer capacity found with each of the two algorithms. For the SOA, the buffer capacity is the optimal total buffer capacity of the MILP model (Weiss & Stolletz, 2015); instead, for the row-column generation, the buffer capacity is an approximation of the total buffer capacity found from the optimal time-buffers as indicated in Alfieri and Matta (2012).

According to Figure 8, the algorithm of SOA is faster than the row-column generation in most of the cases. Specifically, for target cycle time greater than 282, the SOA takes a few seconds to solve the problem, whereas the row-column generation takes from 2 to 3 hours. However, with low μ^*, the row-column generation still takes from 2 to 4 hours to solve the problem, whereas the SOA takes from 20 to 35 hours. The reason of such an increase of computation time is given by the optimal buffer capacity found by the algorithm, which is lower than 50 with larger target cycle time, and from 55 to 77 with lower target cycle time. To find the optimal solution, the SOA decomposes the line in several sub-systems to find bounds, which are used together with the combinatorial cuts to

solve the problem. This procedure is longer as the buffer capacity increases (Weiss & Stolletz, 2015), and Figure 8 confirms it. Instead, the row column generation seems to be buffer capacity invariant, as the computation time is quite stable with different values of μ^*. The row-column generation depends on the size of the problem in terms of number of jobs and stages (as shown in Section 4.1), which are constant in this experiment; thus, the algorithm takes almost the same amount of time to solve all the instances. About the goodness of the approximation of the buffer capacity found by the row-column generation, it seems to over-estimate the optimal buffer capacity (given by the algorithm of Weiss and Stolletz (2015)). However, the trade-off between the computation time and the goodness of the solution should be considered: 29 more hours are needed to find that the total optimal capacity is 77 instead of 91. This means that, in a line of 8 stages, the over-estimation per stage is less than 2 over an optimal average value of about 10.

5 Conclusions

In this paper, a row-column generation algorithm to solve the approximated Buffer Allocation Problem in transfer lines and assembly/disassembly has been proposed. The algorithm extends the results and the modeling framework presented in the previous papers (Alfieri & Matta, 2012; Alfieri et al., 2016). It has been tested against ILOG CLEX on randomly generated instances. The numerical results showed that the row-column generation outperforms CPLEX in more than 87% of the instances and that the difference in the computation times increases as the number of jobs and the number of stages increase. This result shows that the row-column generation is an interesting approach for simulation-optimization problems represented by a single mathematical model. Thus, the same solution approach could be exploited in the recently developed DEO (Descrete Event Optimisation) framework (Pedrielli, Matta, Alfieri, & Zhang, 2018).

ANOVA tests on the solution also showed that the performance of the proposed algorithm is not influenced by the system configuration, i.e., it is immaterial if the system is a transfer line, an assembly system or a disassembly one. This also applies for CPLEX and this is related to the fact that the mathematical model representing the system does not structurally change for the different configurations. Instead, the number of jobs and of stages in the system have the main influence on the computation of the solution.

The proposed algorithm has been also compared to the state-of-the-art exact method to solve the BAP integer problem (Weiss & Stolletz, 2015), in terms of computation times and optimal buffer capacity. The results showed that, although the proposed algorithm is not able to find the optimal buffer capacity, it can be ten times faster than the state-of-the-art method, when a large buffer capacity is needed to achieve the requested throughput. However,

a trade-off must be considered between the quality of the solution and the computation time needed to solve the problem.

Several directions of improvement are still open. First, heuristic procedures can be developed for the search of the initial solution, to reach a feasible one, i.e., to find values of k_0 greater than (but very close to) the optimal index k^*, thus avoiding the possible iterative process due to too small K_{jh} . A second direction of improvement is related to the solution of the Restricted Master Problem (RMP) that, so far, is solved by using the same algorithm the row-column generation is comparing to, i.e., the CPLEX barrier algorithm. Other solution methods, such as sub-gradient procedures or decomposition methods, could be used to speed up its solution when the number of added columns (and hence row) becomes too large. Also, the RMP solution algorithm could be stopped before proving optimality (i.e., once a feasible solution is found for the current set of variables and constraints). A sub-optimal RMP solution, in fact, does not prevent the achievement of global optimality due to the presence of successive iterations, but could reduce the total solution time. Specifically, a sub-optimal solution of the RMP could lead to a situation where more iterations are required, each of which characterized by a smaller computation time. Finally, different methods to find the exact space buffer capacity from the exact time buffer capacity could be explored to have a less approximated solution, thus reducing the importance of the trade-off evaluation between solution quality and computation time.

References

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

Alfieri, A., & Matta, A. (2013). Mathematical programming time-based decomposition algorithm for discrete event simulation. *European Journal of Operational Research*, *231(3)*, 557–566.

Alfieri, A., Matta, A., & Pastore, E. (2016). A column generation algorithm for the buffer allocation problem approximated by the time buffer concept. In *Proceedings of the 8th ifac conference on manufacturing modelling, management and control (mim 2016).*

Altiparmak, F., Dengiz, B., & Bulgak, A. (2007). Buffer allocation and performance modeling in asynchronous assembly system operations: An artificial neural network metamodeling approach. *Applied Soft Computing*, *7(3)*, 946–956.

Andradóttir, S. (1996). A global search method for discrete stochastic optimization. *SIAM Journal on Optimization*, *6*, 513-530.

Banks, J., J.S. Carlson, B. N., & Nicol, D. (2000). *Discrete event systems simulation*. Prentice Hall.

Barnhart, C., E.L.Johnson, Nemhauser, G., Savelsberg, M., & Vance, P. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, *46(3)*, 316-329.

Basu, R. (1977). The interstage buffer storage capacity of non-powered assembly lines a simple mathematical approach. *The International Journal of Production Research*, *15*(4), 365–382.

Buzzacott, J. A., & Shanthikumar, J. G. (1993). *Stochastic models of manufacturing systems*. Prentice-Hall.

Castano, F., Rossia, A., Sevaux, M., & Velasco, N. (2014). A column generation approach to extend lifetime in wireless sensor networks with coverage and connectivity constraints. *Computers & Operations Research*, *52*, 220-230.

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

Dallery, Y., & Gershwin, S. B. (1992). Manufacturing flow line systems: a review of models and analytical results. *Queueing systems theory and applications*, *12(1-2)*, 3-94.

Demir, L., Tunali, S., & Eliiyi, D. (2014). The state of the art on buffer allocation problem: a comprehensive survey. *Journal of Intelligent Manufacturing*, *25*(3), 371–392.

Dolgui, A., Eremeev, A., Kovalyov, M. Y., & Sigaev, V. (2013). Complexity of buffer capacity allocation problems for production lines with unreliable machines. *Journal of Mathematical Modelling and Algorithms in Operations Research*, *12*(2), 155–165.

Dolgui, A., Eremeev, A., & Sygaev, V. (2010, October). A problem of buffer allocation in production lines: complexity analysis and algorithms. *3rd International Conference on Metaheuristics and Nature Inspired Computing*.

Dolgui, A. B., Eremeev, A. V., Kovalyov, M. Y., & Sigaev, V. S. (2018). Complexity of bi-objective buffer allocation problem in systems with simple structure. In *International conference on optimization problems and their applications* (pp. 278–287).

Frangioni, A., & Gendron, B. (2013). A stabilized structured dantzig–wolfe decomposition method. *Mathematical Programming*, *140*(1), 45–76.

Fu, M. (2002). Optimization for simulation: Theory vs. practice. *Journal on Computing*, *14(3)*, 192-215.

Fu, M., Glover, F., & April, J. (2005). Simulation optimization: a review, new developments, and applications. In *Proceedings of the 2005 winter simulation conference.*

Fu, M., & Hu, J. (1997). *Conditional monte carlo: Gradient estimation and optimization applications*. Kluwer Academic.

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

Gershwin, S. (1994). *Manufacturing systems engineering*. Prentice Hall.

Gershwin, S. B. (1987). An efficient decomposition method for the approximate evaluation of tandem queues with finite storage space and blocking. *Operations Research*, *35*, 291-305.

Gershwin, S. B., & Schor, J. E. (2000). Efficient algorithms for buffer space allocation. *Annals of Operational Research*, *93*(1–4), 117–144.

Harris, J., & Powell, S. (1999). An algorithm for optimal buffer placement in reliable serial lines. *IIE Transactions*, *31*, 287-302.

Hemachandra, N., & Eedupuganti, S. (2003). Performance analysis and buffer allocations in some open assembly systems. *Computers & Operations Research*, *30(5)*, 695–704.

Ho, Y., & Cao, X. (1991). *Discrete event dynamics systems and perturbation analysis*. Kluwer Academic.

Ho, Y., Cassandras, C., Chen, C., & Dai, L. (2000). Ordinal optimization and simulation. *Journal of Operations Research Society*, *51*, 490-500.

Hopp, W., & Spearman, M. (2011). *Factory physiscs*. Waveland Pr Inc.

Kleijnen, J. P. C. (1998). Experimental design for sensitivity analysis, optimization, and validation of simulation models. In J. Banks (Ed.), *Handbook of simulation: Principles, methodology, advances, applications, and practice.* John Wiley & Sons.

Kleijnen, J. P. C. (2008). *Design and analysis of simulation experiments* (Vol. 111). Springer.

Kolb, O., & Göttlich, S. (2015). A continuous buffer allocation model using stochastic processes. *European Journal of Operational Research*, *242*(3), 865–874.

Kose, S. Y., & Kilincci, O. (2015). Hybrid approach for buffer allocation in open serial production lines. *Computers & Operations Research*, *60*, 67–78.

Li, J., & Meerkov, S. (2009). *Production systems engineering*. Springer.

Li, L., Qian, Y., Du, K., & Yang, Y. (2016). Analysis of approximately balanced production lines. *International Journal of Production Research*, *54*(3), 647–664.

Lubbecke, M., & Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, *53(6)*, 1007-1023.

Maher, S. J. (2015). Solving the integrated airline recovery problem using column-and-row generation. *Transportation Science*, *50*(1), 216–239.

Martin, G. (1990). Optimal buffer storage capacity in unpaced lines. *Computers & industrial engineering*, *18*(3),

401–405.

Matta, A. (2008). Simulation optimization with mathematical programming representation of discrete event systems. In *Proceedings of the 2008 winter simulation conference.*

Motlagh, M. M., Azimi, P., Amiri, M., & Madraki, G. (2019). An efficient simulation optimization methodology to solve a multi-objective problem in unreliable unbalanced production lines. *Expert Systems with Applications*, *138*, 112836.

Muter, I., Birbil, Ş. İ., & Bülbül, K. (2013). Simultaneous column-and-row generation for large-scale linear programs with column-dependent-rows. *Mathematical Programming*, 1–36.

Myers, R., Montgomery, D., & Anderson-Cook, C. (2009). *Response surface methodology: process and product optimization using designed experiments*. Wiley.

Nahas, N. (2017). Buffer allocation and preventive maintenance optimization in unreliable production lines. *Journal of Intelligent Manufacturing*, *28*(1), 85–93.

Papadopoulos, C., O'Kelly, M., & Tsadiras, A. (2013). A dss for the buffer allocation of production lines based on a comparative evaluation of a set of search algorithms. *International Journal of Production Research*, *51*(14), 4175–4199.

Papadopoulos, C., O'Kelly, M., Vidalis, M., & Spinellis, D. (2009). *Manufacturing systems engineering*. Springer-Verlag New York.

Papadopoulos, H., & Vidalis, M. (1998). Optimal buffer storage allocation in balanced reliable production lines. *International Transactions on Operational Research*, *5(4)*, 325-339.

Pedrielli, G., Matta, A., Alfieri, A., & Zhang, M. (2018). Design and control of manufacturing systems: a discrete event optimization methodology. *International Journal of Production Research*, *56*(1-2), 543–564.

Powell, S., & Pike, D. (1998). Buffering unbalanced assembly systems. *IIE Transactions*, *30*, 55–65.

Renna, P. (2019). Adaptive policy of buffer allocation and preventive maintenance actions in unreliable production lines. *Journal of Industrial Engineering International*, *15*(3), 411–421.

Rubinstein, R. Y., & Shapiro, A. (1993). *Discrete event systems: Sensitivity analysis and stochastic optimization by the score function method*. John Wiley & Sons.

Sadykov, R., & Vanderbeck, F. (2013). Column generation for extended formulations. *EURO Journal on Computational Optimization*, *1*(1-2), 81–115.

Schruben, L. W. (2000). Mathematical programming models of discrete event system dynamics. In *Proceedings of the 2000 winter simulation conference.*

Seo, D.-W., & Lee, H. (2011). Stationary waiting times in m-node tandem queues with production blocking.

IEEE Transactions on Automatic Control, *56*(4), 958–961.

Spall, J. C. (2003). *Introduction to stochastic search and optimization*. Wiley.

Spinellis, D., & Papadopoulos, C. (1998). A simulated annealing approach for buffer allocation in reliable production lines. *Annals of Operations Research*, *93*, 373-384.

Stolletz, R., & Weiss, S. (2013). Buffer allocation using exact linear programming formulations and sampling approaches. *IFAC Proceedings Volumes*, *46*(9), 1435–1440.

Tempelmeier, H. (2003). Practical considerations in the optimization of flow production systems. *International Journal of Production Research*, *41*(1), 149–170.

Tolio, T., & Matta, A. (1998). A method for performance evaluation of automated flow lines. *Annals of CIRP*, *47*(1), 373-376.

Venkateshan, P., & Mathur, K. (2011). An efficient column-generation-based algorithm for solving a pickup-and-delivery problem. *Computers & Operations Research*, *38*, 1647-1655.

Weiss, S., Schwarz, J. A., & Stolletz, R. (2018). The buffer allocation problem in production lines: Formulations, solution methods, and instances. *IISE Transactions*(just-accepted), 1–53.

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

Wolsey, L. (1998). *Integer programming*. Wiley.

Yamada, T., & Matsui, M. (2003). A management design approach to assembly line systems. *International Journal of Production Economics*, *84*, 193–204.

Zabinsky, Z. (2003). *Stochastic adaptive search for global optimization*. Springer.