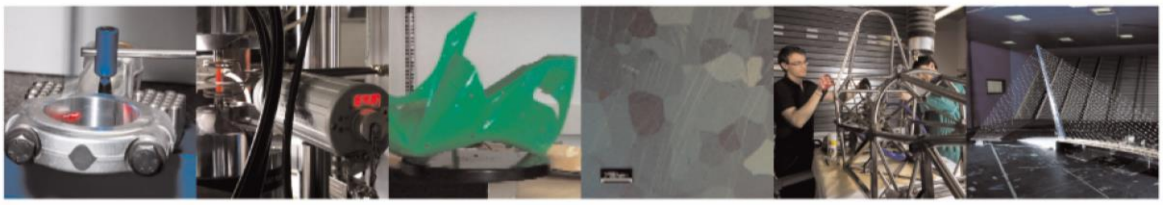


A parallel tabu search for solving the primal buffer allocation problem in serial production systems

Costa, A; Alfieri, Arianna; Matta, Andrea; Fichera, S.

A parallel tabu search for solving the primal buffer allocation problem in serial production systems

A. Costa [a,*], A. Alfieri [b], A. Matta [c], S. Fichera [a]

[a] University of Catania, DII, V.le A. Doria 6, 95125 Catania, Italy
[b] Politecnico di Torino, DIGEP, Corso Duca degli Abruzzi, 24 - 10129 Torino, Italy
[c] Shanghai Jiao Tong University, School of Mechanical Engineering, 800 Dong Chuan Road, Shanghai, China

Available online 9 June 2015

1. Introduction

The buffer allocation problem (BAP) is one of the most challenging issues in the manufacturing system design as it involves several kinds of applications like automatic transfer lines, production/assembly lines, and communication networks. It deals with selecting the optimal storage size to be assigned to every intermediate buffer of a production line. Accordingly to the JIT philosophy, the main role of buffers consists of smoothing and balancing the flow of semi-finished products between two successive stages of a production line, so as to maximize the productivity or to minimize the overall buffer storage capacity. In fact, the bulk of literature on buffer allocation falls into different areas, depending on the goal to be achieved. According to Gershwin and Schor [20], minimizing the total buffer capacity under a minimum production rate constraint represents the primal problem, while the dual problem consists of maximizing the throughput rate of the line without exceeding a fixed amount of buffer size. Furthermore, minimizing the average-work-in-process or maximizing the average profit of the line are two other popular objectives in literature.

In general, to solve a buffer allocation problem, a combination of an evaluative method and a generative method is needed. The former is necessary to assess the throughput rate of the production line for a certain buffer configuration. The latter iteratively makes use of the evaluative method to optimize the buffer configuration.

The buffer allocation problem has been extensively studied in the literature in the last decades as demonstrated by the several surveys [9,4,42,40,15] that describe the multitude of available methodological approaches. The following paragraphs shortly present an overview of these approaches.

Dynamic programming is a popular and effective approach for solving the BAP though the exponential complexity of dynamic programming techniques significantly reduces their applicability to small-sized instances. A pioneer research was conducted by Chow [5], who used the dynamic programming to tackle the buffer allocation problem in reliable production lines. Later, Jensen et al. [26] coped with the problem of optimal location and sizing for buffer inventories in a serial production system that is subject to random variations of timing of stage failures, repair times and demand of finished goods. Huang et al. [25] developed a design method including an approx-imate analytic procedure to evaluate the performance measure of an unreliable production line under a certain allocation strategy. Recently, other contributions adopted the dynamic programming for solving the buffer allocation problem on reliable production lines [16,22].

* Correspondence to: Viale Andrea Doria, 6, 95125 Catania, Italy. Tel.: +39 957382457; fax: +39 95337994.

E-mail address: antonio.costa@dii.unict.it (A. Costa).

Heuristics and search methods, if compared with an exact approach like dynamic programming, generally achieve a near optimal solution in a reasonable computational time, even for large-sized problems. However, due to their restrictive assumptions, their applicability is rather limited [8]. Two heuristic algorithms, namely SEVA and Non-SEVA, based on the concept of local search, have been developed by Seong et al. [48] to maximize the line throughput rate. Jeong and Kim [27] used a conventional gradient search method powered by a sophisticated algorithm to reduce the excessive computation time. Gershwin and Schor [20] analyzed the two well-known problems concerning the BAP, namely the primal and the dual problem. The dual problem is solved by means of a gradient method and the primal problem is faced through another algorithm that makes full use of the dual solution. Solving the dual BAP was the objective of the research conducted by Kim and Lee [29], who proposed a new heuristic algorithm that preliminarily was validated through a numerical comparison with the Non-SEVA algorithm. A significant paper in this field is that by Papadopoulos and Vidalis [43], which presented a heuristic in conjunction with a method able to determine a “good” initial solution that reduces the iterations to find the optimal solution. More recently, an improved local search technique named degraded ceiling algorithm was developed to maximize the throughput rate of unreliable production lines under a given number of total buffer slots [38]. A simple heuristic algorithm is proposed in Hillier and So [24] to estimate the buffer space in a production lines affected by failures, so that the buffer space is able to compensate for the negative effect of failures on the efficiency of the system. The same authors also study how the buffer allocation is affected by the variability of the processing time [23,24].

Metaheuristics play a leading role in solving buffer allocation problems nowadays. They are very popular for their versatility and their search ability combining diversification and intensification strategies. Tabu search, simulated annealing, genetic algorithms and other unconventional algorithms are some of the most popular metaheuristic techniques used for solving the buffer allocation problem. The effectiveness of metaheuristics has been widely demonstrated by the literature over the last years, regard-less of the specific issue to be faced. Nevertheless, a well-known drawback of such techniques is that they must be tailored to the specific problem. To this end, the parameters driving the search mechanism need to be selected through a preliminary calibration analysis. However, such a weakness may be partially compensated by adopting a so-called adaptive structure, according to which the algorithm parameters are automatically adjusted during the optimization path, e.g., on the basis of the current iteration. A first approach to figure out a buffer allocation problem is that of Lutz et al. [31]. The main purpose of their study was to develop a procedure based on a tabu search, combined with simulation, for determining the size and location of work-in-process inventory buffers. Notably, they coped with the primal buffer allocation problem through a tabu search that may employ two distinct search modes, namely Swap and Global search. Each search mode runs the subset of neighbor buffer profiles in a different way. In addition, in order to enhance the performance of their tabu search, the authors implemented a dynamic concept of tabu list, neighborhood set and change factor, in step with the modern trend embracing the adaptive approach. Spinellis and Papadopoulos [52] used both the exact numerical algorithm and the decomposition algorithm to solve the dual buffer allocation problem by means of simulated annealing and genetic algorithms. Dolgoui et al. [17] developed a genetic algorithm where the tentative solutions are evaluated with an approximate method based on the Markov-model aggregation approach. The goal was to select the buffer capacities of an unreliable production line so as to minimize a cost-based objective function. In addition to the serial algorithms above described, there are hybrid and parallel metaheuristics. A hybrid metaheuristic is one that combines a metaheuristic with other optimization approaches, such as algorithms from mathematical programming, constraint programming, and machine learning [56]. A first attempt in terms of hybrid optimization algorithms is that of Shi and Shuli [49], who incorporated the tabu search heuristic into the Nested Partitions framework. The obtained numerical results confirmed the effectiveness of their approach in large production lines. A hybrid metaheuristics for solving the buffer allocation problem in unreliable tandem production lines was developed in (Dolgoui et al. [18]). In particular, they proposed a genetic algorithm combined with a branch-and-bound that exploits the Markov model aggregation technique for goal function evaluation. A genetic algorithm connected with a simulation-based evaluation method and an Artificial Neural Network, was proposed to investigate the buffer allocation of unbalanced – unreliable flow lines [30]. The cross-entropy method is a new paradigm for stochastic optimization that Alon et al. [1] used along with a simulation approach to solve the throughput rate maximization for the BAP. The authors stated that it is able to quickly generate near-optimal solutions for fairly large production lines. To determine the optimal buffer allocation for maximum line throughput rate and maximum line economic profit an artificial immune system has been recently presented [34]. Also, it is worthy to point out that they made use of a decomposition method to address the aforementioned issue related to unreliable production lines. Two research contributions emphasizing the effectiveness of tabu search in solving the BAP are ascribable to [13,14]. First, they presented an adaptive tabu search (ATS) for designing unreliable non-homogenous production lines with the objective of throughput rate maximization. Then, they proposed a binary search (BS) and a tabu search (TS) for solving the primal problem aiming to minimize the total buffer size under through-put rate constraint. Numerical results revealed that there is not a significant difference of performance between the two optimiza-tion techniques, whereas a lower computational time is required by BS on the average. Recently, five metaheuristics, also including a tabu search, have been compared in order to detect which search algorithm is more suitable to find an efficient solution of the BAP [41]. From their experiments, simulated annealing and a myopic heuristic search resulted the most performing algorithms, whereas tabu search performed quite well for small and medium flow lines. The measures of performance were CPU time and closeness to the maximum throughput rate.

The goal of this paper is to introduce a new metaheuristic algorithm able to solve the buffer allocation problem in a more efficient and effective way than the other metaheuristics proposed by literature so far. A parallel tabu search (PTS) algorithm has been developed to address the primal buffer allocation problem in reliable production lines. Differently from the previous tabu search procedures, as for example those proposed by Demir et al. [14] and Lutz et al. [31], PTS exploits the parallel configuration inherited from the parallel computing field to enhance the search strategy over the available space of solutions. In addition, an adaptive neighborhood generation mechanism that tunes the entity of buffer size variations according to the number of iterations allows a quick convergence of the parallel branches towards feasible and promising near-optimal solutions. Then, PTS holds a proper restart procedure that strengthens the intensification strategy and avoids to be trapped into local optima. Although neither physical nor virtual parallel computing architectures have been used for the proposed PTS, the combination of the parallel structure composed by three different branches with a properly designed tight neighborhood allows to reduce the number of iterations needed for convergence and to favor the algorithm efficiency. In order to validate the effectiveness of PTS, an extended comparison analysis, involving four metaheuristics and an exact method, has been arranged. Since the buffer allocation problem at hand can be classified as a non-linear integer problem, three real-encoding metaheuristics usually adopted to address non-linear and even non-differentiable objective functions have been derived from the literature on applied mathematics and computation. The fourth method consists of a binary search recently presented in the literature specifically for addressing the BAP. All the metaheuristic algorithms employ the same simulation-oriented evaluative procedure to measure the throughput rate of a given buffer configuration. As for the exact optimization method, it consists of a MILP-based simulation/optimization approach derived from [35]. Due to the exponential complexity of the problem, it was used to generate the optimal solutions related to small- and medium-sized problems.

The remainder of the paper is organized as follows. The problem statement is reported in Section 2. Section 3 deals with all the optimization approaches adopted in this paper, namely linear programming-based simulation/optimization, parallel tabu search, genetic algorithm, differential evolution, particle swarm optimization and binary search. Section 4 explains how the numerical examples have been generated. In particular, the different factors involved in the benchmark generation are examined. Section 5 includes both the multiple-comparison statistical analysis and the computational analysis, which emphasize the effectiveness of the proposed optimization approach. Finally, the findings from of a multiple-factor ANOVA analysis along with a series of main effect and interaction plots have been discussed in the last sub-section. Section 6 concludes the paper.

2. Description of the problem

We consider the buffer allocation problem in a reliable serial production line with J stages and $J-1$ buffers, on which N identical parts with stochastic processing times have to be processed. The processing policy is a-priori known so that no scheduling problem has to be addressed. No machine breakdown may occur and the first machine is never starved. The release time of each part and the processing time of each part on each stage are known in advance and are denoted as a_i $(i=1,\ldots,N)$ and t_{ij} $(j=1,\ldots J)$, respectively. In particular, stochastic processing times are assumed to be drawn from a general distribution and, due to the absence of scheduling decisions, $a_i \le a_{i+1}$ for each part i. Completion time of part i at stage j is denoted by y_{ij}. Each stage j, with exception of the first, is preceded by a buffer with finite capacity $B_j \in [0, N-1]$ $(j=1, \ldots J-1)$, while the first stage is supposed to be preceded by a buffer with infinity capacity. Part i needs to be temporarily stored in buffer B_j whether stage $j+1$ is processing another part. The objective is minimizing the total buffer capacity subject to the constraint that the corresponding throughput rate must be greater than or equal to a lower bound $\theta*$ fixed by the system designer.

Basically, if Z is the total buffer capacity to be allocated into the ($J-1$) intermediate buffers, the total amount of possible allocations increases dramatically with Z and J, according to Eq. (1):

$$\left.\begin{matrix} Z+J-2 \\ J-2 \end{matrix}\right) = \frac{(Z+1)(Z+2)\cdot\cdot\cdot(Z+J-2)}{(J-2)!} \tag{1}$$

3. Solving the stochastic buffer allocation problem

Among the evaluative methods proposed in the literature, the most popular are Markov state [43,17,18], decomposition method [19] and simulation [31,35,47]. Basically, Markov models are not practical to assess the throughput rate of a production line with finite buffers. Indeed, due to the exponential growth in the number of states, the exact computation of the production rate for lines with more than two machines is a tricky task. Hence, most of the evaluation methods adopted to analyze such issues consider either analytic approximation or simulation. Analytical approximation methods, with the exception of Markov models for two-machine-one buffer models, are based on either aggregation or decomposition approaches. On the other hand, simulation models based on discrete-event models are computationally time-expensive but allow an exact analysis of the system performance and they can be adapted to several kinds of problems. However, both approximation and simulation models make use of system states that, in case of lines with multiple machines and buffers, tends to become very large [51].

In this paper, an alternative evaluative method based on a recursive procedure has been devised. After the randomness of part processing times is realized by random sampling, the pro-posed procedure exactly computes the throughput rate of a production line once the size of each buffer has been fixed. Similarly to what is done by the discrete-event simulators, such method accurately replicates the behavior of the system in a time-effective way. The simplicity of the algorithm encourages both repeatability and easiness of implementation in every develop-ment framework, so that no discrete-event simulation package is required. In this research it was implemented through the visual basic editor within a regular MS Excel® worksheet. The VBA® code of the proposed algorithm is reported in Appendix A.

As far as the generative method is concerned, the proposed PTS and three additional metaheuristics using the same evaluative algorithm will be presented in detail in the following. Since exact solutions represent a valid reference to validate both efficacy and efficiency of any metaheuristic method, first we deal with the exact simulation/optimization model adopted to optimally solve a set of small- and medium-sized benchmarked problems.

3.1. Mathematical programming formulation

Whenever the effectiveness of a meta-heuristic technique needs to be evaluated, exact solutions may represent a valid reference to be compared with. To this end, this section describes the mixed integer linear programming (MILP) model derived from [35], which both simulates the flow line and selects the optimal capacity to be allocated to each buffer.

$$\min Z = \sum_{k=L_j}^{U_j} z_{jk} k \tag{2}$$

s.t.

$$y_{i1} \ge a_i + t_{i1}, \forall i = 1, \ldots, N \tag{3}$$

$$y_{i+1,j} - y_{ij} \ge t_{i+1,j}, \quad \forall i = 1, \ldots, N-1, j = 1, \ldots, J-1 \tag{4}$$

$$y_{i,j+1} - y_{ij} \ge t_{i,j+1}, \quad \forall i = 1, \ldots, N, j = 1, \ldots, J-1 \tag{5}$$

$$y_{i+k,j} - y_{i,j+1} \ge t_{i+k,j} z_{jk} - M(1 - z_{jk}), \quad \forall j = 1, \ldots, J-1, \\ i = 1, \ldots, N-k, k = L_j, \ldots, U_j \tag{6}$$

$$\sum_{k=L_j}^{U_j} z_{jk} = 1, \quad \forall j = 1, \ldots, J-1 \tag{7}$$

$$\frac{N}{y_{NJ}} \ge \theta* \tag{8}$$

where, the objective function (2) is the minimization of the total buffer capacity Z. z_{jk} is a binary variable equal to one if capacity of buffer B_j is equal to k ($k=L_j,\dots,U_j$); L_j and U_j are the bounds for the j-th buffer defined by the decision maker. Constraints (3) simply provide that the completion time of a part in the first machine must be greater than or equal to its arrival time plus its processing time. Constraints (4) state that a machine cannot process two consecutive parts at the same time. Constraints (5) impose that a part cannot be processed by the downstream machine if it is occupied by another part. Constraints (6) make use of the *big M* to manage the following condition. If $z_{jk}=1$ and k is the capacity of B_j between stage j and $j+1$, then the completion time of part i on stage $j+1$ must be lower than or equal to the starting time of part $i+k$ on stage j. Only one capacity k must be chosen for each buffer B_j as imposed by constraints (7). According to constraint (8) the mean throughput rate must be greater than or equal to a minimum value θ^*.

In reality, the generalized approach provides that L_j and U_j should be equal to 1 and $N-1$, respectively. In this research, the output from the metaheuristics has been used to quantify U_j for speeding up the convergence of the MILP model, as follows: $U_j=2\times \max_j(B_j^*)$, where B_j^* is the capacity of the j-th buffer in the best solution among the five metaheuristics.

3.2. The parallel tabu search

Buffer allocation problem is a *NP*-hard combinatorial optimization problem [25,32]. Two different approaches can be adopted to tackle such kind of issue: exact methods and metaheuristics.

In this paper, a novel tabu search named Parallel Tabu Search (PTS) has been developed. Tabu Search is a trajectory based metaheuristics algorithm for addressing combinatorial optimization problems ([3]). It was first introduced by Glover [21] and basically consists of a neighborhood search method that keeps track of the up-to-now search path to avoid to be trapped into local optima or to try an explorative search of the solution domain. Unlike the regular single-trajectory tabu search, the framework of PTS is inspired by the field of parallelism in computer science. A parallel metaheuristic uses the techniques of parallel programming to run multiple metaheuristic searches in parallel; these may range from simple distributed schemes to concurrent search runs that interact to improve the overall solution [56]. In fact, differently form the regular tabu search, which iteratively makes use of a single seed to generate the corresponding neighborhood to investigate the space of solutions, the proposed technique exploits the search ability of three parallel seeds that periodically exchange information and interact each other through a common optimization framework. Although the proposed PTS has been designed as a sequential metaheuristics, i.e., with no parallel distribution of the computational burden, the structure of the proposed PTS allows to be executed in large clusters or grids equipped with GPUs (Graphics Processing Units) and multi-core processors, so as to significantly increase the quality of solution as well as to decrease the computational time. In fact, the construction of parallel algorithms makes it possible to increment significantly the number of solutions considered (in a unit of time) using multi-processor computing environments.

Fig. 1 shows the flow chart of the proposed PTS; lighter boxes refer to the parallelized processes, darker boxes should be executed on the main framework. The pseudo-code of PTS is also reported. In addition to the neighborhood generation procedure, that will be hereinafter discussed, two distinct procedures have been properly developed to enhance the search mechanism connected to the buffer allocation problem, namely Initial Seed Generation (ISG) and Stochastic Buffer Distribution (SBD), respectively. SBD procedure supports another phase of the optimization procedure named seed re-generation, which replaces the currents seed solutions in case of no improvement after $N_{max}/5$ evaluations occurs. The same seed re-generation is applied whether it has never been applied before (*flag*=0); it means that a new local optimum has been found just before the algorithm stop. Consequently, since a further investigation of the space of solutions is needed, the number of evaluations to stop the algorithm is increased to $1.3\times N_{max}$, i.e., $0.3\times N_{max}$ additional evaluations are executed. A detailed description of the aforementioned procedures is reported on the following.

Procedure: parallel tabu search (PTS)

```
Step 1:   Initialization: $flag=0$, $eval=0$, $c\_worse=0$, $Nmax$;
Step 2:   launch ISG procedure and elaborate the total
          buffer capacity$B_{tot}$;
Step 3:   generate the three seed solutions, $S1$, $S2$, $S3$. $S1$ is
          generated making full use of the ISG procedure.
          The other two seeds are generated as follows:
          $B_k(S2)=\lceil B_{tot}/(n-1)\rceil, \quad \forall k\in 1,\dots,n-1;$
          $B_k(S3)=\lceil B_{tot}/[2\times(n-1)]\rceil, \quad \forall k\in 1,\dots,n-1;$
Step 4:   generate the neighborhood for each seed;
Step 5:   evaluate objective function for each solution of
          the neighborhoods. Select the new candidate
          seed ($S1'$, $S2'$, $S3'$) accordingly to both the
          aspiration criterion and the tabu list;
Step 6:   select the best seed ($S\_best$) among $S1'$, $S2'$, $S3'$, i.
          e, the seed that provides the best objective
          function.
Step 7:   update the number of evaluated solutions ($eval$).
          If $eval>N_{max}$ then two options may occur: (a) if
          $flag=0$ (i.e., no seed regeneration has been
          carried so far) increase $N_{max}$, flag$\leftarrow$1, goto Step
          11, (b) if $flag=1$ stop the PTS; else, go to Step 8;
Step 8:   if $S\_best$ improve the current local optimum then
          store $S\_best$ and restore the counter of worse
          solutions $w\_count$; then goto Step 10; else,
          update the worse solutions counter and goto
          Step 9;
Step 9:   if $w\_count<N_{max}$ goto Step 10, else goto Step 11;
Step 10:  replace the old seeds with the new ones and goto
          Step 4;
Step 11:  keep track that one seed regeneration has been
          performed, at least. Make use of the SBD
          procedure applied three times to $S\_best$ to
          generate three new seeds, $S1'$, $S2'$, $S3'$; goto
          Step 4;
```

3.2.1. Initial seed generation (ISG)

Generally, the first seed solution from which the search mechanism starts plays a key role for a successful application of metaheuristics and tabu search as well. In most literature about metaheuristics applied to BAP, the designer arbitrarily fixes an initial total buffer capacity sensibly lower than $(J-1)\times(n-1)$, regardless of any rationale or quantitative method (See for example [14,18,57]). As a result, the search ability of any metaheuristics may be strongly supported by such a forced choice and the global performance may be significantly improved.

In this paper, a specific procedure named IGS has been developed to overcome any uncertainty in the choice of the initial buffer capacity, as it aims to generate an initial seed solution assuring the maximum throughput rate. The IGS procedure is reported hereinafter and works as follows.

First, a maximum capacity $N-1$ to each buffer ($B_k=N-1$ $\forall k=1,\dots,J-1$) is assigned as to obtain an infinite-buffered flow line whose production rate is equal to θ_{max}. Successively, the whole

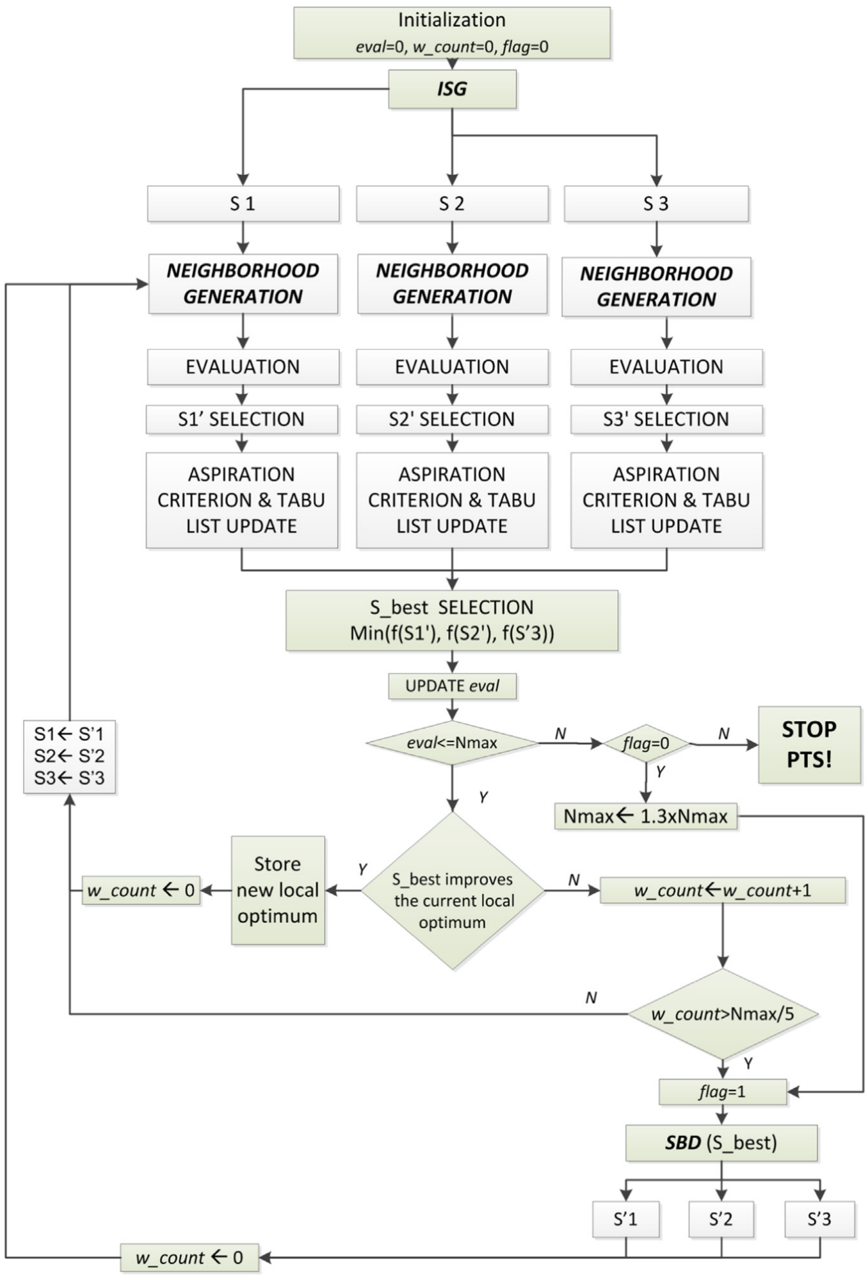


Fig. 1. PTS flow chart.

set of parts is scheduled through the flow line by means of the evaluative algorithm (see Appendix A) and the buffer capacity actually required by each buffer is a-posteriori computed through the algorithm portion reported in Step 3 (See IGS procedure). Since the obtained buffer configuration still assures the maximum throughput rate θ_{max}, it may be considered as a feasible initial solution for the proposed buffer allocation problem and the related total buffer capacity B_{tot} can be assumed as an upper bound for the problem at hand.

Hence, the IGS procedure makes it possible to launch any metaheuristics from a feasible buffer allocation whose total capacity is significantly lower than the maximum allowable one, i.e., $(J-1)\times(N-1)$. Consequently, the space of feasible solutions can be favorably restricted and the expected computational times reduced too.

Procedure: Initial Seed Generation (IGS)

```
Step 1: initialization;
$i=1,\dots,N$; part index;
$j=1,\dots,J$;machine index;
$B_k=N-1, \quad \forall k=1,\dots,J-1$; temporary buffer size;
$\overline{B}_k=0, \quad \forall k=1,\dots,J-1$; final buffer size;
Step 2: schedule jobs through the infinite-buffer flow line.
  Compute $t_{i,j}$, $\overline{t}_{i,j}$;
Step 3: compute the actual capacity $\overline{B}_j$ of each buffer;
$B_k=0, \quad \forall k=1,\dots,J-1$
for $j=2$ to $J$
  for $i=1$ to $N$
    for $k=i$ to $N-1$
    if $\overline{t}_{k+1,j-1}<t_{i,j}$ and $\overline{t}_{k+1,j-1}>t_{i,j-1}$ then
    $B_{j-1}\leftarrow B_{j-1}+1$
      if $B_{j-1}>\overline{B}_{j-1}$ then
```

```
            $\overline{B}_{j-1} \leftarrow B_{j-1}$
            **endif**
          **else**
          **goto** Step 4
          **endif**
  **next** $k$
  *Step 4*: continue
          $B_{j-1} \leftarrow 0$
     **next** $i$
  **next** $j$
  $B_{tot} = Sum_k(\overline{B}_k)$
```

3.2.2. *Stochastic buffer distribution*

In order to improve the exploration ability of PTS, a proper Stochastic Buffer Distribution (SBD) procedure has been devised. To avoid a premature convergence to a local optimum, it stochastically re-allocate the total buffer capacity B_{tot} related to the current best solution. Basically, such procedure works similarly to a roulette wheel criterion. First, the average processing times per each stage are computed. Then, they are normalized and cumulated with respect to the number of production stages. Hence, for each unit of buffer capacity to be allocated, a random number x is drawn from a uniform distribution U[0,1]. Of course, the bigger is the average processing time of a given stage, the greater is the probability of assigning a unit of capacity to the corresponding upstream buffer. As reported in **Fig. 1**, the SBD procedure is enabled whether there is not any objective function improvement after $N_{max}/5$ evaluations. The pseudo-code of such procedure is as follows.

Procedure: Stochastic Buffer Distribution

```
Step 1:   $B_{tot}$=total buffer capacity connected to the current
          solution;
Step 2:   considering the whole set of $N$ parts to be
          processed, compute the average processing time
          per each machine;
Step 3:   normalize the average processing times;
Step 4:   cumulate the normalized values and generate a
          probability interval for each buffer;
Step 5:   set $b=1$;
Step 6:   draw $x \in U[0,1]$;
          select the probability interval and the
          corresponding buffer where such unit of buffer
          capacity has to be allocated;
          $b \leftarrow b+1$;
Step 7:   **if** $b < B_{tot}$ **then** goto Step 6 **else** goto Step 8;
Step 8:   stop the procedure.
```

3.2.3. *Move representation and adaptive neighborhood generation*

A neighborhood structure is a mechanism which gains a new set of neighbor solutions by applying a small perturbation to a given seed solution. Each neighbor solution is reached immediately from a given seed solution by a move [21]. Defining the neighborhood of a given seed solution is a crucial phase of the TS development as its structure may considerably affect the search ability through the search space and the computational efficiency as well. Therefore, unnecessary and infeasible moves must be eliminated if it is possible [39]. Generally, the structure of the neighborhood from a seed solution depends on the kind of problem to be tackled. For instance, in the buffer allocation problem, all neighbors of a given seed may be taken into account or, in alternative, only a subset of those may be considered. In this paper, a neighborhood structure made by two distinct sub-structures has been devised. Each sub-structure depends on a specific move and, as a result, the whole neighborhood entails $2 \times (J-1)$ solutions. Each move is denoted by the notation $[\eta,\nu]$ where η is the amount of capacity added/subtracted to element, i.e. buffer, ν of the seed solution. In fact, η may assume either positive or negative value, thus generating the two mentioned sub-structures, each one populated by $(J-1)$ neighbors. The first sub-structure is connected to a negative value of η $(-\eta)$ and holds the neighbors from 1 to $J-1$. If S is the seed solution and $B(S)_1$ is the buffer capacity related to the first location then the first neighbor S^1 is identical to S with exception of the first buffer whose capacity will be $B(S^1)_1 = B(S)_1 - \eta$. In the same way, the second neighbor S^2 is identical to S with exception of the second location that will be $B(S^2)_2 = B(S)_2 - \eta$, and so on. The second sub-structure refers to the positive value of η $(+\eta)$ and includes the neighbors from J to $2 \times (J-1)$. Similarly as before, the first neighbor is identical to the seed S except for $B(S^J)_1 = B(S)_1 + \eta$, the second neighbor except for $B(S^{J+1})_2 = B(S)_2 + \eta$, and so on. In this manner, each solution of the neighborhood is univocally identified. If the move $[\eta,\nu]$ generates a better objective function than the concurrent move then $[-\eta,\nu]$ is not permitted. i.e., it is a tabu move. Table 1 shows an example of neighborhood for the seed solution $S=(7-8-15-4-12)$ for a buffer allocation problem with 6 machines and 5 buffers. Underlined locations denote the only difference between each neighbor and the seed solution. The role of the first sub-structure (hereinafter called $-\eta$-neighborhood) is quite clear. It aims to minimize the total buffer capacity to be allocated. Conversely, the second sub-neighborhood (hereinafter called $+\eta$-neighborhood) is crucial when the current seed is a local optimum and all its $-\eta$ neighbors are infeasible, i.e., they do not satisfy the throughput rate related constraint. In this case, since every infeasible solution is strongly penalized by the objective function, a non-tabu solution included in the $+\eta$-neighborhood can be accepted as a new seed. In this way, the combination of such twofold neighborhood structure with the proposed objective function ensures an effective balance between exploration and exploitation phases.

Exploitation and exploration, or intensification and diversification, are two major components of any metaheuristic algorithms, ([3]). Diversification means to generate diverse solutions so as to explore the search space on a global scale, while intensification means to focus the search in a local region knowing that a current good solution is found in this region. A good combination of these two components will usually ensure that global optimality is achievable.

The first part of the optimization path concerning any metaheuristic algorithm consists in an extended exploration phase, according to which several areas of the search space are quickly investigated and the quality of the initial solution may be drastically improved. Adopting a unit of buffer capacity as a move to generate the neighborhood could excessively slow down such initial exploration phase, thus increasing the computational time of the whole tabu search. In order to speed up the diversification phase, an adaptive configuration of the neighborhood generation mechanism has been devised. It is based on a proper exponential function that adaptively changes the η value according to the current iteration. Notably, the role of such function is to assign a high value to η during the first part of the search, thus enhancing the diversification strategy. The higher is the number of iterations, the smaller will be the η value, i.e., the unit of buffer capacity to be

Table 1
Example of neighborhood for the seed $S=(7-8-15-4-12)$.

μ	Neighbor	ν				
		1	**2**	**3**	**4**	**5**
-1	*1*	6	8	15	4	12
	2	7	7	15	4	12
	3	7	8	14	4	12
	4	7	8	15	3	12
	5	7	8	15	4	11
$+1$	*6*	8	8	15	4	12
	7	7	9	15	4	12
	8	7	8	16	4	12
	9	7	8	15	5	12
	10	7	8	15	4	13

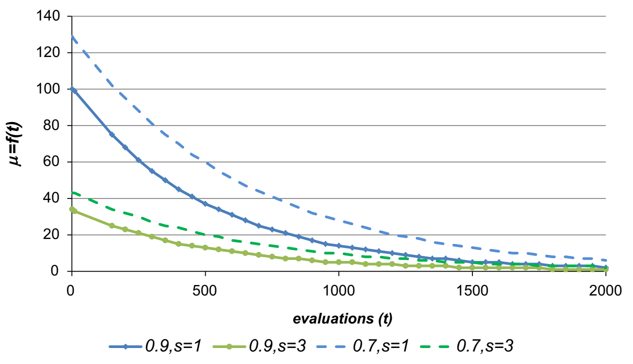


Fig. 2. $\mu=f(t)$ graphs as α and s change.

allocated. Then, once η is equal to one, a regular intensification phase can take place. The following function has been considered where B_{max} is the maximum value of buffer capacity in a certain seed, α and β are two specific parameters and t is the amount of currently evaluated solutions.

Symbol $\lfloor\rfloor$ indicates that the quantity in parenthesis must be rounded down to the nearest integer.

$$\eta=\left\lfloor \frac{B_{\max}}{s\cdot\gamma\cdot\beta}\times\exp\left(\frac{-t\times 100}{N_{\max}}\times\frac{1}{\alpha}\right)\right\rfloor+1 \tag{9}$$

Using the exponential function to adaptively control variables/ parameters of metaheuristics is a common practice in adaptive optimization. Such methodology has the effect of narrowing the search, and gradually changes the model from an exploratory mode to an exploitative mode.

Parameter γ was fixed to $\theta^*/0.9\theta_{\max}$. This value is motivated by the fact that the smaller is θ^* the smaller will be the required total buffer capacity; thus, if $\theta^*<0.9\theta_{max}$ the convergence speed towards lower values of buffer capacity increases, the provided number of iterations being equal. In turn, since seed $S3$ initially holds a lower total buffer capacity, it was necessary to adequate η to the related smaller solution space; thus, parameter s must be set to 3 only for seed S3 and to 1 for the other seed solutions, $S1$ and $S2$.

To illustrate how the exponential function works, Fig. 2 shows the $\eta=f(t)$ diagram for a value of B_{max} equal to 200. As the number of evaluations t increases, the η value adaptively decreases until a unit buffer capacity is achieved. Continuous curves refer to $\theta^*=0.9\theta_{max}$, i.e., $\gamma=1$, while dotted lines relate to $\theta^*=0.7\theta_{max}$, i.e., $\gamma<1$. The two lower curves show the variation of μ for seed $S3$ ($s=3$), while the upper curves relate to the η variation concerning with either seed $S1$ or seed $S2$ ($s=1$). Moreover, it should be noted that a faster convergence of η to one occurs when $\theta^*=0.9\theta_{max}$, s being equal. As far as parameter s is concerned, if it is equal to one a stronger algorithm diversification is allowed; otherwise, $s=3$ encourages the algorithm intensification in the neighborhood of seed $S3$.

After a proper calibration, parameters α and β were fixed to 4 and 2, respectively. The structure of the aforementioned function as well as the other parameters have been designed through a series of trial-and-error runs on several instances with 9 and 15 machines, whose results are not reported for the sake of brevity.

3.2.4. *Further characteristics of PTS*

As mentioned before, tabu moves allow TS to escape from local optima. Nevertheless, good solutions could be rejected because of the same tabu moves. As a result, it is necessary to provide a so-called aspiration criterion to disregard a tabu move in certain situations. A well-established aspiration criterion consists of allowing a move, even though it is tabu, whether it is able to improve the current best solution. All the three branches of the proposed PTS have been equipped with such aspiration criterion.

The termination criterion of the PTS consists of the total number of solutions evaluated N_{max}. As discussed in the previous section, the N_{max} threshold may be increased to 1,3N_{max} during the search path whether the regeneration procedure has been never used before, i.e., when a new local optimum has been found just before the algorithm stop. The size of the tabu list has been fixed to 7, in accordance with most TS applications [45].

3.2.5. *Objective function*

Over the last few decades, several methods have been proposed to handle constraints in the evolutionary algorithms. A common approach consists of the use of penalties [6,50,59].

Several penalty methods may be used to transform a constrained problem into an unconstrained one. In this paper we consider an objective function composed of two functions (See Eq. (10)). The upper one aims to minimize the total buffer capacity while maximizing the production rate. The lower function penalizes solutions whose throughput rate is lower than θ^*. It is based on a static approach, as the penalty applied to infeasible solutions do not depend on the current generation number.

$$f(B_{tot},\theta)=\begin{cases} B_{tot}+\frac{1}{c\vartheta},\text{if}\,\theta\ge\theta^* \\ K-(B_{tot}+\vartheta),\text{otherwise}\end{cases} \tag{10}$$

Parameter c is necessary to classify two close feasible solutions. In words, it allows to privilege a certain solution over another one which holds an extra unit of buffer capacity and a significantly higher throughput rate. For example, let suppose the former solution is characterized by $B_{tot}=100$ and $\theta_1=0.35$ while the latter provides $B_{tot}=101$ and $\theta_2=0.81$. To assure the priority of the total buffer capacity over the throughput rate it must be:

$$c>\frac{\Delta\theta}{1-\Delta\theta} \tag{11}$$

where $\Delta\theta$ is equal to $\theta_2-\theta_1$. Thus, only a value of c greater than 1.04 is able to assure the correct ranking between the aforementioned solutions. In this study, c was set to 2 to support a difference on the throughput rate up to 0.667.

As for the penalty function, K is a large positive constant whose only restriction is that it must guarantee any infeasible solution is graded worse than a feasible one. In this study K was fixed to $(n-1)\times(J-1)$. Also, the structure of this penalty function properly fits the problem at hand. In fact, since the throughput rate is an increasing function of the total buffer capacity, this function aims to privilege infeasible solutions with higher total buffer capacity and throughput rate, as they have a higher probability to become feasible throughout the successive generations.

Usually, a penalization is more efficient if its expression is related to the amount of constraint violation than to the violated constraint number [46]. Since in this study only one constraint needs to be handled and the throughput rate is an increasing function of the total buffer size, the measure of violation may be considered the throughput rate itself whenever it is lower than θ^*.

3.3. Other generative approaches

In order to validate the efficiency of the proposed PTS the following metaheuristics have been implemented. A Binary Search (BS), recently presented by the relevant literature to cope with the primal buffer allocation problem, and three evolutionary techniques from the field of applied mathematics and computer science, namely a Genetic Algorithm (GA), a Particle Swarm Optimization (PSO) and a Differential Evolution (DE).

As for the proposed evolutionary algorithms, they do not make use of any additional procedure or local search focusing on the buffer allocation problem. The search path starts from a population generated by means of the ISG procedure and makes use of regular perturbation operators, not specifically oriented to the problem at hand.

Conversely, the binary search was specifically developed to solve the total buffer capacity minimization problem under a minimum throughput rate. In the following, the difference between the general purpose metaheuristics and the focused optimization technique will be investigated under both the efficacy and the efficiency viewpoint.

3.3.1. Genetic algorithm

Genetic Algorithms (GA) are stochastic optimization techniques that exploit a population based heuristic methodology. They belong to the class of evolutionary algorithms (EA) which mimic the principle of survival of the fittest individuals by Charles Darwin. GAs, like most metaheuristics, need an encoding scheme to represent the problem in terms of a numerical string named chromosome; thus, a chromosome holds the decision variables of the problem and, as a consequence, it is a solution of the problem itself. A set of chromosomes are arranged into a population and then, repeatedly, a series of generations are executed making full use of the following major operators: selection, crossover and mutation. Recently, real coded genetic algorithm has been used to solve integer and mixed integer optimization problems [11].

In this paper, a real-coded genetic algorithm has been implemented for solving the stochastic buffer allocation problem. It consists of a regular real coded GA that was inspired to the HX-NUM configuration presented by Deep and Thakur [12]. Population has been randomly generated and each chromosome is coded as a string of $J-1$ integer values; each gene refers to the capacity of the corresponding buffer and it is defined in the range $[1, N-1]$.
Chromosomes are selected according to a Tournament method before being subject to the heuristic cross-over operator. Actually both the Laplace and the Heuristic cross-over operator have been tested on a set of preliminary runs that involved different sizes of problem. For the sake of brevity, numerical results have not been reported in this paper. However, the final outcome was the clear outperformance of the GA equipped with the heuristic cross-over. Then, a regular non-uniform mutation [36] was embedded into the proposed real-coded GA. Finally, it is worth pointing out that an elitism strategy, which preserves the fittest individual during the different generations, has been included in the genetic framework. Integrality of variables was ensured by rounding real values down to the nearest integer. If a gene exceeds the provided upper bound due to a perturbation operator, then it is forced to be equal to the upper bound itself, i.e., $N-1$. On the other hand, if a gene assumes a negative value then it is forced to be equal to the lower bound, i.e., zero. The same strategy has been used for the other real-coding metaheuristic algorithms.

As for the genetic parameters, population size has been set to 50, the tournament size is equal to 2, while both probability of crossover and mutation has been selected after a calibration analysis whose results will be illustrated in Section 5.1. The stopping criterion is the total number of generations.

3.3.2. Differential evolution

During the last decades Differential Evolution (DE) [55,54] became one of the most popular evolutionary computation methods, able to solve continuous optimization problems. There is probably a twofold reason to explain such success. Firstly, the efficacy of DE in solving benchmark, engineering and real-world problems, as confirmed by the literature [10,44]. Secondly, in comparison with other recently developed metaheuristics, the basic DE is very easy to be understood as well as simple to be implemented. Conforming to the well-established nomenclature scheme, a *DE/rand/1/bin* version of DE has been implemented according to the procedure reported in Appendix B.

As for the setting parameters, population size has been set to $M=20$ while parameters F and CR have been selected after a calibration analysis whose results will be illustrated in Section 5.1. Since buffer sizes must be integers, during the algorithm evolution real values assumed by variables are rounded to the nearest integer. The stopping criterion depends on the total number of iterations.

3.3.3. Particle swarm optimization

Particle Swarm Optimization (PSO) is a modern evolutionary technique introduced by James Kennedy and Heberhart [28]. It consists of a metaheuristic algorithm based on the social metaphor and gained attention as an effective tool for solving various optimization problems. Similarly to genetic algorithms, PSO is a stochastic, population-based optimization technique. Conversely, it is easier to be implemented and does not require a significance computational burden. Due to its algorithm structure, PSO allows to carry out a diversification phase at the beginning of the search and a local search at the end, according to the provided inertia weight parameter and acceleration coefficients, respectively. Unlike the canonical PSO that uses a large inertia weight and static acceleration coefficients, namely cognitive and social parameters, the proposed PSO is inspired to the GLBest version [2], where the aforementioned parameters are adaptively managed in terms of global best and local best values. For the sake of clarity, the equations of the adopted GLBest algorithm are illustrated in Appendix C.

Both inertia weight w_i (Eq. (17)) and acceleration coefficient c_i (Eq. (18)) depend on two parameters, λ and μ. The authors of GLBest fixed them to 1.1 and 1, respectively. In this paper, they have been selected through a proper calibration analysis. Similarly to the previous metaheuristics, the initial swarm has been randomly generated by means of the ISG procedure. The swarm dimension has been fixed to 20 particles. Similarly to the previous techniques, the algorithm stops when a given number of iterations have been achieved.

3.3.4. Binary search

The Binary Search (BS) considered in this paper consists of a hybrid meta-heuristic algorithm incorporating a tabu search algorithm to manage the throughput rate related constraint. The algorithm structure is identical to the BS proposed by Demir et al. [14], the only difference being the initial solution generation mechanism that is based on the proposed ISG method. Actually,

Table 2
Factors for the selection of parameters.

Metaheuristics	PSO		GA		DE	
Parameter	λ	μ	p_{cr}	p_m	F	CR
High (H)	1.9	2	0.8	0.05	0.90	0.95
Low (L)	1.1	1	0.5	0.01	0.65	0.30

the aforementioned authors used BS as an alternative technique to validate a novel tabu search (TS). However, since the computational experiments highlighted a balance of performance between the two approaches and revealed the outperformance of BS in terms of computation time, it was preferred to TS. It is worth reminding that the original BS was developed to cope with a buffer allocation problem in unreliable production lines where the total buffer capacity to be allocated ranges from 50 to 400, on the basis of the number of machines. The numerical outcomes revealed that the performance of both BS and TS is strongly affected by the choice of the initial capacity to be allocated, as much as the number of machines increases. Accordingly to these findings, since the choice of the initial buffer capacity has not been supported by any rigorous criterion so far, the implemented BS was equipped with the IGS method, so as to align all the proposed metaheuristics under the starting solution viewpoint.

Due to the complexity of the overall optimization procedure, which embeds an adaptive tabu search into the binary search, the structure of the BS algorithm has been here omitted. For further details readers should refer to [13,14].

4. Computational experiments

A comprehensive experimental campaign has been carried out to strengthen the effectiveness of the proposed parallel tabu search with respect to the other metaheuristics. To this end, an extended benchmark of test problems has been generated on the basis of four different factors: number of production stages (i.e., machines), type of line, desired throughput rate, and coefficient of variation of processing times. Different levels have been provided for each factor, as follows:

- Production stages (PS): $5-10-15$;
- Type of line (TL): balanced (BAL), unbalanced with bottleneck in the first station (UNB_F), unbalanced with bottleneck in the middle station (UNB_M), unbalanced with bottleneck in the last station (UNN_L);
- Throughput rate (TR): High (H), i.e., 90% of the maximum achievable θ_{max}; Low (L), i.e., 70% of the maximum achievable θ_{max};
- Coefficient of variation (CV): High (H), i.e., $\sigma^2/\mu^2=1.5$; Low (L), i.e., $\sigma^2/\mu^2=0.5$, where μ and σ^2 are mean and variance of the processing time distribution, respectively.

A number of parts equal to 2000 has been considered to simulate the problem at hand. For each combination of the aforementioned parameters, 10 instances have been generated. For each instance, processing times have been extracted from a *log-normal* distribution. If the line is balanced then the mean is set to 2 and the variance arises from the selected CV value (High/Low). Otherwise, if the type of line is unbalanced then processing times of the bottleneck station are extracted from a *log-normal* distribution with mean equal to 4 while, as for the other machines, the mean is set to 2. Again, the variance depends on the selected CV level. For each instance, θ_{max} has been computed by means of the evaluative procedure reported in Appendix A assuming $B_k=J-1\,\forall k$, i.e., simulating an infinite-buffered production system. In conclusion, the total amount of runs provided for each metaheuristics is equal to $3\times4\times2\times2\times10=480$. Since 5 distinct metaheuristics have been compared and 5 replicates connected to different random seeds have been run for each instance, the total amount of experiments to compare such metaheuristics is equal about to $5\times2400=12{,}000$. Furthermore, both small-(5 machines) and medium-sized (10–15 machines) problems have been optimally solved by means of the MILP-based simulation/optimization approach; thus additional 320 runs were executed. No warm-up period was taken into account as it is demonstrated that no statistically significant difference exists between test simulations with and without a warm-up [53]. The proposed metaheuristics have been implemented on the VBA® development framework, within a MS Excel® worksheet executed on a PC equipped with an Intel® Core duo 2, 33 GHz processor and 2 Gb RAM memory.

5. Numerical results

Several experimental analyses have been carried out to support the validation of the proposed parallel metaheuristic procedure. Firstly, since the effectiveness of evolutionary algorithms notoriously depends on how their working parameters are set, a proper tuning analysis has been performed. Then, the selection of parameters α and β, characterizing the adaptive neighborhood generation mechanism of PTS, has been investigated through the same statistical approach. Once the best configuration of parameters has been chosen for each metaheuristics, a multiple-comparison statistical analysis has been developed to identify the most performing optimization technique. Finally, on the basis of the numerical results obtained by the best metaheuristics, the statistic influence of factors mentioned in Section 4, namely PS, TL, TR, CV, has been studied through a proper ANOVA analysis.

5.1. Calibration of the evolutionary algorithms

Three one-way statistical analyses have been arranged in order to select the more suitable parameter configuration for each evolutionary technique: PSO, GA, DE. Two factors, corresponding to two distinct parameters, have been considered as independent variables, for each metaheuristics. Each factor is varied to two-levels, namely High (H) and LOW (L), as a set of preliminary test on the medium-sized problems allowed to select the most two promising levels. As a result, four different configurations have been compared. Table 2 reports the parameters involved in the tuning analysis for each optimization technique.

The benchmark problems used for the calibration analysis is the same descripted in Section 4, the only difference consisting of the number of instances (five instead of ten), the number of seed-based replications (1 instead of 5) and the number of parts to be processed (1000 instead of 2000). Hence, a total amount of $3\times5\times2\times2\times2\times4=480$ runs selecting the best parameter configuration of each algorithm. For the calibration phase, a different random benchmark has been employed in terms of processing times, to avoid any overfilling issues and biased results. The total buffer capacity was considered as the response variable while the following Relative Percentage Deviation (RPD) function was considered to both normalize and compare the obtained results

$$RPD=\frac{RV_p-RV_{best}}{RV_{best}} \tag{12}$$

where RV is the value of the response variable related to a given instance, which adopts a given parameter configuration p, while

```
Kruskal-Wallis Test on RPD_PSO
 FACTORS      N      Median    Ave Rank      Z
 LL         240  0.000000000      499.7   1.24
 HL         240  0.000000000      426.0  -3.52
 LH         240  0.000000000      500.9   1.32
 HH         240  0.000000000      495.5   0.97
 Overall    960                   480.5
 H = 12.42  DF = 3  P = 0.006
 H = 18.44  DF = 3  P = 0.000  (adjusted for ties)

 Kruskal-Wallis Test on RPD_GA
 FACTORS      N      Median     Ave Rank      Z
 LL         240  0.062500000      550.0   4.48
 HL         240  0.038520000      534.5   3.49
 LH         240  0.000000000      439.4  -2.65
 HH         240  0.000000000      398.1  -5.32
 Overall    960                   480.5
 H = 50.65  DF = 3  P = 0.000
 H = 59.35  DF = 3  P = 0.000  (adjusted for ties)

 Kruskal-Wallis Test on RPD_DE
 FACTORS      N      Median     Ave Rank      Z
 LL         240  0.000000000      468.9  -0.75
 HL         240  0.000000000      510.6   1.94
 LH         240  0.000000000      490.7   0.66
 HH         240  0.000000000      451.8  -1.85
 Overall    960                   480.5
 H = 6.14  DF = 3  P = 0.105
 H = 9.34  DF = 3  P = 0.025  (adjusted for ties)
```

Fig. 3. Kruskall – Wallis tests to calibrate PSO, GA and DE.

```
Kruskal-Wallis Test on RPD

 CONFIG      N       Median    Ave Rank      Z
 LL        240  0.000000000      484.0   0.22
 HL        240  0.000000000      478.0  -0.16
 LH        240  0.000000000      482.1   0.10
 HH        240  0.000000000      478.0  -0.16
 Overall   960                   480.5
 H = 0.09  DF = 3  P = 0.994
 H = 1.63  DF = 3  P = 0.652  (adjusted for ties)
```

Fig. 4. Non-parametric tests for the PTS calibration.

RV_{best} is the best value achieved by one of the four alternative configurations, i.e., $RV_{best} = \min(RV_p)\ \forall p = 1, 2\ldots, 4$.

In order to find the best parameter configuration for each metaheuristics, three statistical analyses, using the RPD measure as response variable, have been arranged. First, the three main assumptions of ANOVA were checked: normality, homoscedasticity and independence [37]. Since no group of the mentioned RPDs satisfied the normality test, a non-parametric test was required as data are gathered into a lower bound, i.e., the differences between the optimum values of the benchmarks and the returned values are gathered into 0_+.

A Kruskal – Wallis (K – W) test [7] has been employed as it may be considered as the non-parametric counterpart of a one-way ANOVA and it may be used to test for a difference between the medians of 3 or more groups of data. If the p-value is lower than 0.05, the result is significant, which means that there is a significant difference between at least two of the medians.

Three distinct H tests have been implemented by MINITAB®14 to calibrate each metaheuristics. Since two parameters have to be tuned for each algorithm, four groups characterize each test. The outcomes of each K – W test are reported in Fig. 3 and all of them indicate that there is a statistically significant difference between the obtained results at a 95% confidence level. The p-values (adjusted for ties) arising from each test for PSO, GA and DE are equal to 0.000, 0.000 and 0.025, respectively. These results confirm that the parameter configurations statistically affect the performance of every metaheuristic algorithm. Since such non-parametric test does not specify which configuration is the best, the following considerations about the obtained rank scores and the medians have been carried out. As for PSO, all the medians are equal to zero, regardless of the specific combination of parameters, but the Z rank connected to group HL, which employs $\lambda = 1.9$ and $\mu = 1.0$, gains the only negative value, thus indicating that there is a significant difference between this configuration and the others.

The output results from the K – W test on the GA calibration denote a difference about both medians and Z scores of the provided groups. As a result, the last two configurations, namely LH and HH, outperform the others. Due to a lower Z rank the HH parameter setting, which involves $p_m = 0.8$ and $p_{cr} = 0.05$, has been selected for the genetic algorithm. As far as the DE is concerned, the medians computed by the adopted statistical test are the same for each group. The negative Z scores related to LH and HH highlight the most performing configurations but the latter group involving $F = 0.90$ and $\mathrm{CR} = 0.95$ has been definitely selected because of the lower Z score.

5.2. Tuning analysis of PTS

As discussed in Section 3.2.3, the proposed parallel tabu search makes full use of an adaptive neighbor generation function whose role is speeding up the diversification phase. It assumes a value of μ that gradually decreases during the search path, according to a negative-exponential function depending on the number of iterations as well as on two specific parameters, namely α and β. Similarly to the previous metaheuristics, a proper calibration study has been performed on two levels (High (H), Low (L)) of parameters α and β. As for α, such two levels have been set to 4 and 2, respectively; whereas, 5 and 2 have been chosen for β. The benchmark of test cases used to calibrate the PTS is identical to that used for the tuning analysis of the other evolutionary algorithms. The obtained outcomes revealed a significant robustness of the PTS, as most of the RPD values, properly computed to compare the different configurations, were equal to zero. As a result, the group of RPDs cannot satisfy the hypothesis of normality, and a non-parametric statistical analysis was required. A Kruskal – Walls test has been carried out to investigate any statistical difference between the different parameter configurations. Fig. 4 reports the output of the non-parametric test from MINITAB®14. The corresponding p-value equal to 0.652 demonstrates as the null hypothesis cannot be rejected, i.e., there is not any significantly statistical difference among the medians of the groups. However, due to the lower Z score obtained by two of the groups, configuration HL, namely $\alpha = 4$ and $\beta = 2$, has been selected for the subsequent experiments.

5.3. Multiple-comparison of metaheuristics

In this section a comprehensive comparison analysis involving the proposed metaheuristic algorithms is presented. They are assessed in terms of both quality of solutions and computational times. First, three tables at varying number of machines have been presented. Each table holds two sets of data: the relative percentage deviation ($\Delta\%$) between each metaheuristics and the best returned value, and the number of wins (w), which indicates how many times a given algorithm reaches the best objective. Numerical results are hierarchically sorted according to the following factors: type of line (TL), coefficient of variation (CV) and throughput rate (TR), respectively. Actually, numerical results reported in each table refer to the average values computed over the 10 instances and the five seed replicates provided by the adopted benchmark. For each sub-group of results associated to every line type, the average $\Delta\%$ and wins are computed, for each metaheuristics. The last row of each table denotes the grand average values concerning with both $\Delta\%$ and wins. As for small- and medium-sized problems involving 5 and 9 machines, respectively, it worth

Table 3
Average numerical results for the 5 machine flow line.

TL	CV	TR	Δ%_PSO	Δ%_GA	Δ%_DE	Δ%_PTS	Δ%_BS	w_PSO	w_GA	w_DE	w_PTS	w_BS
BAL	H	H	0.001	0.055	0.000	0.000	0.000	10	4	10	10	10
	H	L	0.000	0.036	0.000	0.000	0.000	10	10	10	10	10
	L	H	0.000	0.025	0.000	0.000	0.000	10	9	10	10	10
	L	L	0.000	0.000	0.000	0.000	0.000	10	10	10	10	10
ave/sum			*0.000*	*0.029*	*0.000*	*0.000*	*0.000*	*40*	*33*	*40*	*40*	*40*
UNB_F	H	H	0.023	0.068	0.000	0.000	0.000	10	10	10	10	10
	H	L	0.045	0.045	0.003	0.000	0.000	10	10	10	10	10
	L	H	0.000	0.009	0.000	0.000	0.000	10	10	10	10	10
	L	L	0.000	0.000	0.000	0.000	0.000	10	10	10	10	10
ave/sum			*0.017*	*0.030*	*0.001*	*0.000*	*0.000*	*40*	*40*	*40*	*40*	*40*
UNB_M	H	H	0.349	0.087	0.202	0.000	0.000	9	10	9	10	10
	H	L	0.095	0.048	0.028	0.000	0.000	9	9	10	10	10
	L	H	0.063	0.028	0.050	0.000	0.000	8	10	7	10	10
	L	L	0.000	0.010	0.000	0.000	0.000	10	10	10	10	10
ave/sum			*0.127*	*0.043*	*0.070*	*0.000*	*0.000*	*36*	*39*	*36*	*40*	*40*
UNB_L	H	H	0.478	0.129	0.151	0.000	0.000	7	9	10	10	10
	H	L	0.257	0.108	0.133	0.000	0.000	9	9	10	10	10
	L	H	0.117	0.037	0.031	0.000	0.000	10	10	10	10	10
	L	L	0.000	0.016	0.000	0.000	0.000	10	10	10	10	10
ave/sum			*0.213*	*0.073*	*0.079*	*0.000*	*0.000*	*36*	*38*	*40*	*40*	*40*
g_ave			**0.089**	**0.044**	**0.037**	**0.000**	**0.000**	**38.0**	**37.5**	**39.0**	**40.0**	**40.0**

pointing out that the $\Delta\%$ performance indicator is computed on the basis of the global optimum gained by solving the MILP-based simulation/optimization model. Conversely, the $\Delta\%$ values related to the scenario problem with 15 machines refer to the minimum buffer capacity over the five metaheuristics to be compared and with respect to the five seed replicates for each optimization procedure. Table 3 shows that both PTS and BS always reach the global optimum, regardless of the specific type of flow line. On the other hand, both PSO and DE perform quite well when the line type is balanced or unbalanced at the first stage (UNB_F), while their effectiveness slightly decreases in the other scenario problems, namely UNB_M and UNB_L, particularly when both CV and TR assume the higher value. PSO is more effective then GA in case of BAL and UNB_F line types. Nevertheless, GA on the average outperforms PSO as the former seems to be less influenced by the line type and by the other factors as well.

As for medium-sized problems, Table 4 highlights again the outperformance of PTS and BS with respect to the other competitors. In fact, both of them ensure a grand average result almost equal to zero, which means that in most test cases they achieve the global optimum. In addition, looking at the number of wins, PTS confirms its effectiveness as always there exists at least one replication out of five that is able to reach the global optimum, beyond the specific combination of factors. Although the DE algorithm significantly improves its performance, similarly to the previous analysis, it seems negatively affected by a kind of line unbalanced either in the middle or in the last stage. Nevertheless, DE outperforms BS in terms of average number of wins. Finally, PSO and GA reveal a deterioration of performance as demonstrated by the average $\Delta\%$ as well as by the average number of wins equal to 27 and 16 out of 40, respectively.

Table 5 reports the numerical results related to the large-sized problems. The main finding from this table concerns the downgrading performance of BS, likely due to the premature stop enabled whenever a time limit equal to 16,200 s, i.e., four hours, is exceeded. In fact, the higher computational burden required by the larger number of variables negatively affects the search ability of the BS algorithm. The obtained results strengthen the effectiveness of the proposed PTS optimization procedure, even though DE reaches a similar outcome also under the number of wins viewpoint. As expected, both PSO and GA do not take any advantage of the large sized problems, which further emphasize their weakness in solving the buffer allocation problem.

In order to statistically infer if any algorithm performs better than others, a proper investigation analysis has been carried out. Due to the global effectiveness of the tested algorithms, most of obtained results in terms of RPD are equal to zero, so that data are skewed and cannot be classified as normally distributed. As a result, a conventional parametric test like ANOVA cannot be used as the main hypotheses of normality and homogeneity of variance surely are not satisfied. Therefore, in order to avoid any meaningless result, similarly to what has been done for the calibration issue, a Kruskal–Wallis non-parametric technique has been employed to compare the performance of the different algorithms. Fig. 5 reports the output of the test, performed by MINITAB® 14. The final p-value equal to 0.000 confirms a significant difference between at least two algorithms. The medians as well as the Z rank put in evidence the outperformance of PTS, DE, and BS with respect to GA and PSO. In addition, according to the obtained ranking results, PTS is recommended as the most performing optimization algorithm. Although the returned values are not normally distributed, Fig. 6 allows evaluating such findings. It shows the 95% confidence plot intervals for the means and, in addition, it holds a series of green marks representing the corresponding median values. Three algorithms out of five, PTS, BS and DE, have the same median. However, Fig. 6 puts in evidence the outperformance of PTS and DE with respect to BS, as their variance is very tight and no overlap there exists with the other plot intervals. To investigate more in detail the difference of performance between PTS and DE, a post-hoc pairwise comparison based on a non-parametric Mann–Whitney U test [33] has been executed. Fig. 7 shows the output from MINITAB® 14, which proves a statistically significant difference ($p = 0.000$) in the median engagement between PTS and DE with 95% CI. Hence, the outperformance of PTS is ascertained.

5.4. *Computational comparison and convergence analysis*

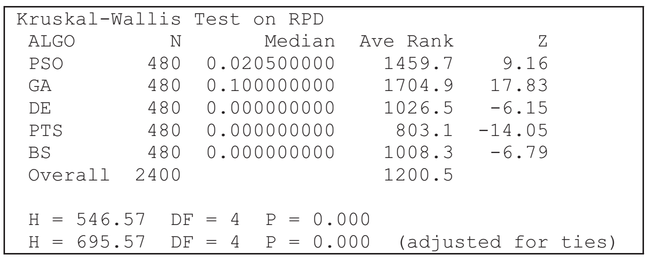
Beyond the efficacy in terms of quality of solutions, the computational efficiency of a metaheuristics is a leading factor to be evaluated. As mentioned in the earlier sections, the stopping criterion of each metaheuristics is the total number of iterations (*iter*) or objective function evaluations. Table 6 depicts the total

Table 4
Average numerical results for the 9 machine flow line.

TL	CV	TR	Δ%_PSO	Δ%_GA	Δ%_DE	Δ%_PTS	Δ%_BS	w_PSO	w_GA	w_DE	w_PTS	w_BS
BAL	H	H	0.004	0.187	0.000	0.000	0.000	9	0	10	10	10
	H	L	0.014	0.357	0.002	0.002	0.011	4	0	10	10	5
	B	H	0.004	0.055	0.002	0.000	0.000	10	9	10	10	10
	B	L	0.003	0.222	0.000	0.000	0.000	10	0	10	10	10
ave/sum			*0.006*	*0.205*	*0.001*	*0.000*	*0.003*	*33*	*9*	*40*	*40*	*35*
UNB_F	H	H	0.123	0.157	0.003	0.000	0.000	4	1	10	10	10
	H	L	0.069	0.168	0.001	0.000	0.000	9	2	9	10	10
	B	H	0.000	0.000	0.000	0.000	0.000	10	10	10	10	10
	B	L	0.007	0.074	0.000	0.000	0.000	10	7	10	10	10
ave/sum			*0.245*	*0.100*	*0.001*	*0.000*	*0.000*	*33*	*20*	*39*	*40*	*40*
UNB_M	H	H	0.324	0.183	0.042	0.001	0.000	2	0	9	10	10
	H	L	0.441	0.301	0.035	0.000	0.000	5	0	10	10	10
	B	H	0.004	0.024	0.002	0.000	0.000	10	10	10	10	10
	B	L	0.138	0.124	0.020	0.003	0.000	3	6	10	10	10
ave/sum			*0.227*	*0.158*	*0.025*	*0.001*	*0.000*	*20*	*16*	*39*	*40*	*40*
UNB_L	H	H	0.397	0.198	0.043	0.000	0.000	0	1	10	10	10
	H	L	0.486	0.308	0.019	0.000	0.000	6	1	10	10	10
	B	H	0.000	0.004	0.000	0.000	0.000	10	10	10	10	10
	B	L	0.084	0.076	0.009	0.002	0.000	6	7	10	10	10
ave/sum			*0.242*	*0.147*	*0.018*	*0.002*	*0.000*	*22*	*19*	*40*	*40*	*40*
g_ave			**0.131**	**0.152**	**0.011**	**0.000**	**0.001**	**27.0**	**16.0**	**39.5**	**40.0**	**38.8**

Table 5
Average numerical results for the 15 machine flow line.

TL	CV	TR	Δ%_PSO	Δ%_GA	Δ%_DE	Δ%_PTS	Δ%_BS	w_PSO	w_GA	w_DE	w_PTS	w_BS
BAL	H	H	0.021	0.555	0.004	0.002	0.045	2	0	7	10	0
	H	L	0.034	0.361	0.001	0.001	0.103	6	0	10	10	5
	B	H	0.033	0.516	0.001	0.000	0.026	6	0	10	10	5
	B	L	0.053	0.319	0.002	0.000	0.010	7	2	10	10	8
ave/sum			*0.035*	*0.438*	*0.002*	*0.001*	*0.046*	*21*	*2*	*37*	*40*	*18*
UNB_F	H	H	0.102	0.472	0.003	0.001	0.016	5	0	10	10	5
	H	L	0.226	0.239	0.002	0.007	0.034	0	0	10	10	4
	B	H	0.021	0.182	0.001	0.001	0.000	9	0	10	10	10
	B	L	0.000	0.000	0.000	0.000	0.000	10	10	10	10	10
ave/sum			*0.084*	*0.223*	*0.002*	*0.002*	*0.013*	*24*	*10*	*40*	*40*	*29*
UNB_M	H	H	0.375	0.425	0.003	0.000	0.028	3	1	10	10	6
	H	L	0.343	0.329	0.020	0.011	0.626	1	1	10	10	1
	B	H	0.127	0.183	0.008	0.005	0.789	1	0	10	10	0
	B	L	0.000	0.005	0.000	0.000	1.769	10	10	10	10	0
ave/sum			*0.211*	*0.236*	*0.008*	*0.004*	*0.803*	*15*	*12*	*40*	*40*	*7*
UNB_L	H	H	0.555	0.483	0.011	0.008	0.110	1	0	10	10	3
	H	L	0.503	0.352	0.018	0.009	0.861	0	0	9	10	2
	B	H	0.107	0.143	0.005	0.000	0.694	3	5	10	10	2
	B	L	0.000	0.003	0.000	0.000	1.627	10	10	10	10	2
ave/sum			*0.291*	*0.245*	*0.008*	*0.004*	*0.823*	*14*	*15*	*39*	*40*	*9*
g_ave			**0.156**	**0.285**	**0.006**	**0.003**	**0.421**	**18.5**	**9.8**	**39.0**	**40.0**	**15.8**



```
Kruskal-Wallis Test on RPD
 ALGO         N        Median  Ave Rank        Z
 PSO        480  0.020500000    1459.7     9.16
 GA         480  0.100000000    1704.9    17.83
 DE         480  0.000000000    1026.5    -6.15
 PTS        480  0.000000000     803.1   -14.05
 BS         480  0.000000000    1008.3    -6.79
 Overall   2400                 1200.5

 H = 546.57  DF = 4   P = 0.000
 H = 695.57  DF = 4   P = 0.000   (adjusted for ties)
```

Fig. 5. Comparison of metaheuristics: Kruskal – Wallis non-parametric test. Output from Minitab 14®.

number of iterations provided per each algorithm, as the number of machines changes. The corresponding number of total objective functions evaluations (*eval*) has been reported in italics. As for PTS, the stopping criterion is based on the total number of evaluated

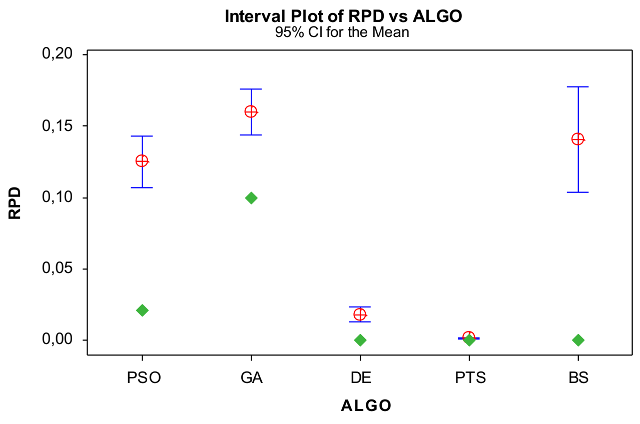


Fig. 6. Comparison of metaheuristics: plot interval 95% confidence.

```
Mann-Whitney Test and CI: PTS; DE
        N  Median
 PTS  480  0.0000
 DE   480  0.0000

 Point estimate for ETA1-ETA2 is -0.0000
 95.0 Percent CI for ETA1-ETA2 is (-0.0001;0.0000)
 W = 204197.5
 Test of ETA1 = ETA2 vs ETA1 not = ETA2 is significant at 0.0000
 The test is significant at 0.0000 (adjusted for ties)
```

Fig. 7. Comparison between PTS and DE: Mann – Whitney U test output.

Table 6
Comparison of metaheuristics: total iterations and evaluations provided for each metaheuristics.

Machines		5	9	15
GA	iter	200	300	400
	eval	*8000*	*14000*	*18000*
PSO	iter	400	700	900
	eval	*8000*	*14000*	*18000*
DE	iter	400	700	900
	eval	*8000*	*14000*	*18000*
PTS	*eval*	*5000*	*10000*	*15000*
	max eval	*(6500)*	*(13000)*	*(18000)*

Table 7
Comparison of metaheuristics: average computational times (s).

Machines	PSO	GA	DE	PTS	BS	LP
5	45.1	41	42.2	30.5	102.0	766.7
9	172.4	149.5	169.6	122.7	3532.8	2994.9
15	382.6	342.9	375.9	340.7	8478.2	–

```
Analysis of Variance for RESP, using Adjusted SS for Tests

Source     DF    Seq SS    Adj SS   Adj MS        F      P
Blocks      9      3655      3655      406     0,98  0,454
A           2    419805    419805   209903   506,78  0,000
B           3   1046493   1046493   348831   842,20  0,000
C           1      7480      7480     7480    18,06  0,000
D           1    566077    566077   566077  1366,71  0,000
E           4         2         2        1     0,00  1,000
A*B         6    179762    179762    29960    72,33  0,000
A*C         2    396132    396132   198066   478,20  0,000
A*D         2     94616     94616    47308   114,22  0,000
A*E         8         9         9        1     0,00  1,000
B*C         3       642       642      214     0,52  0,671
B*D         3    436361    436361   145454   351,18  0,000
B*E        12         5         5        0     0,00  1,000
C*D         1       738       738      738     1,78  0,182
C*E         4         3         3        1     0,00  1,000
D*E         4         2         2        1     0,00  1,000
Error    2334    966720    966720      414
Total    2399   4118505

S = 20,3517   R-Sq = 76,53%   R-Sq(adj) = 75,87%
```

Fig. 8. PTS analysis: ANOVA table from MINITAB®14.

solutions and the maximum value that may be reached when extra evaluations are allowed is indicated in parenthesis. Due to the structure of the BS algorithm, no limit may be fixed in terms of either generation or evaluations. Therefore, it has been implemented according to the same setting used in the literature (See [14]). In order to emphasize the computational effectiveness of PTS, the maximum number of evaluations has been deliberately reduced with respect to the other competitors.

Table 7 reports the average computational times per each technique as the size of the flow line changes. Since the MILP programming (LP) based optimization was able to solve only small- and medium-sized test problems in a reasonable time, Table 6 misses of the result corresponding to the large-sized problems involving 15 machines. Numerical results put in evidence the efficiency of PTS under the computational viewpoint, even though the difference with the other algorithms in terms of average time to converge decreases as the problem size increases. Specifically for the large-sized issue, such a kind of finding is motivated by the adaptive behavior of PTS that is able to improve the maximum number of allowed iterations whether any solution improvement occurs just before the algorithm stop. In light of the aforementioned remarks, BS and LP can be considered as comparable in terms of average computational times, though it is necessary to remind that the latter approach assures the optimality of solutions.

Among the other evolutionary algorithms, the Differential Evolution may represent a valid alternative to PTS as confirmed by the quality of the solutions and the average computational times. Its structure consists of a regular DE algorithm; thus, its performance could be likely improved by introducing a hybridizing local search algorithm or by devising an alternative version of mutation operator that better fits the BAP at hand. Finally, both PSO and GA do not seem to be suitable to address the proposed buffer allocation problem.

5.5. *ANOVA analysis for PTS*

The last step of the numerical study consisted in a sensitivity analysis about the several influencing factors considered in the proposed research. On the basis of the numerical results provided by PTS, a multi-factor model experimental design has been carried out to evaluate how the number of machines, the type of line, the coefficient of variation and the minimum allowed throughput rate influence the response variable, i.e., the total buffer capacity. The random blocking factor "Blocks" coincides with the problem and is varied at 10 levels, that is, for each class 10 different instances are generated. The fixed factor A is varied at 3 levels and coincides with the number of machines the flow line is composed of, namely 5, 9 and 15 machines. B is the fixed factor concerning with the type of line and it varies at 4 levels: BIL, UNB_F, UNB_M, UNB_L. The fixed factor C refers to the minimum throughput rate to be satisfied: it is varied at two levels (1 = Low, 2 = High). The fixed factor D involves the coefficient of variation of the lognormal distribution from which the processing times have been drawn and is varied at two levels (1 = High, 2 = Low), in reverse to factor C. The seed of the generative algorithm is the random factor E, which is varied at 5 levels: for each instance the algorithm is run five times, each corresponding to a different random evolution. Totally, $3 \times 4 \times 2 \times 2 \times 10 \times 5 = 2400$ runs have been executed to complete the experimental plan. The influence of this last random factor has been investigated with the aim of concluding that the obtained optimal buffer capacities are independent from the random seed. Given the structure of the experimental plan, interactions among blocks and factors were a-priori excluded. After the main hypotheses of ANOVA have been preliminarily checked, a proper analysis of variance has been carried out through the General Linear Model routine of Minitab®14. Fig. 8 illustrates the ANOVA table including terms in the model up through order two, which indicates, as expected, that all the factors are significant, with exception of the Block as well as of factor E. The statistically significant influence of factor A on the response variable is trivial, while the influence of the other factors along with some factor interactions merits a deeper investigation.

Fig. 9 shows the main effect plots of factors B, C, D and E. The influence of factor B is clearly motivated by the upper-left plot, which demonstrates that the total buffer capacity of the balanced flow lines is higher than the unbalanced ones. The factor C related plot shows that the limit on the throughput rate has not a strong influence on the total buffer capacity to be allocated; in fact, the significance of factor C is featured by the lower Fisher value *F* among the other influencing factors. Conversely, the main *F*-value

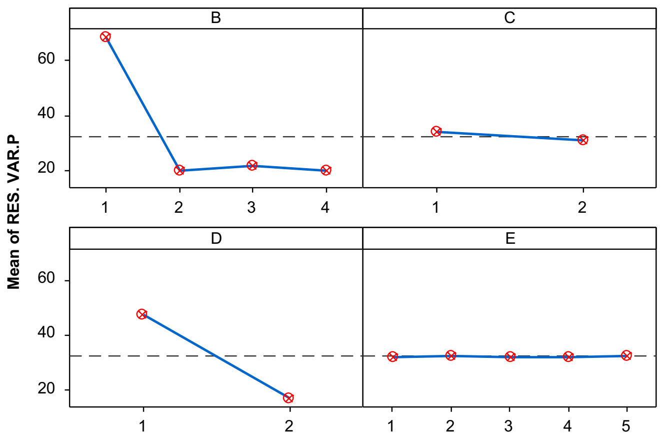


Fig. 9. PTS analysis: main effect plot for factors B, C, D, E.

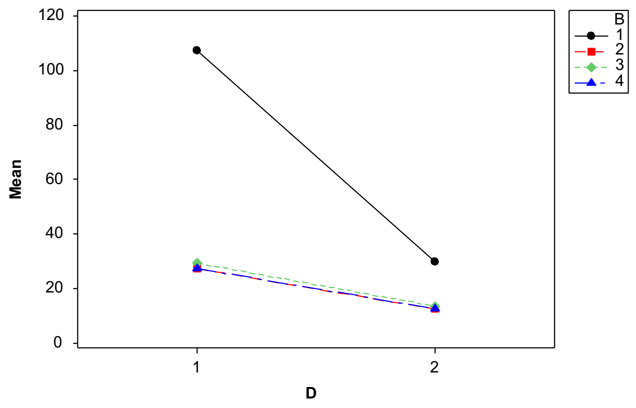


Fig. 10. PTS analysis: interaction plot (data means) for the response variable.

is achieved by factor D, namely the coefficient of variation, as confirmed by the slope of the main effect plot. The sensitivity of the model to this factor is an expected result and, as a result, the higher is the coefficient of variation the higher is the total buffer capacity to be allocated. Finally, the lower-right plot diagram confirms that the random seed is not a significant factor of the model, thus emphasizing the robustness of the proposed PTS algorithm. Regardless of the interactions involving factor A, which clearly conditions the other factors, it is worth investigating the interaction between factor B and D, which is graphically represented by the interaction plot of Fig. 10. The main finding consists of the different slope between the balanced line related plot and the other plots. The balanced flow line is significantly influenced by the coefficient of variation, while the unbalanced ones seem to be insensitive to the change of coefficient of variation.

6. Conclusions

In this paper, the optimization of the primal buffer allocation problem through metaheuristic algorithms has been investigated. A parallel tabu search, PTS, which makes full use of an adaptive neighborhood generation mechanism has been presented. PTS algorithm was compared to three real-encoding metaheuristics derived from the relevant literature on the field of applied mathematics and computation. Then, a binary search specifically developed in the literature for addressing the primal buffer allocation problem has been implemented. A comprehensive set of benchmark problems have been generated according to several experimental factors and the optimal solution of both small- and medium-sized instances have been computed by means of a MILP-based simulation/optimization approach. An exact evaluative method consisting of a recursive algorithm has been devised to simulate the behavior of the system. The main parameters of the real-encoding metaheuristic algorithms and PTS have been selected on the basis of a proper non-parametric calibration analysis. Then, a multiple-comparison statistical test, embracing all the provided optimization techniques, has been carried out to individuate the most performing algorithm.

The obtained numerical results confirmed the outperformance of PTS with respect to the other procedures. The binary search incorporating an adaptive tabu search demonstrated a good performance for small and medium sized instances, even though the high computational time it requires to converge severely limits its application. A significant sensitivity to the initial total buffer capacity to be allocated represents another drawback of this method. The differential evolution algorithm confirmed its effectiveness as well as its computational efficiency, despite the basic configuration. In fact, the narrow gap from PTS, regardless of the size of the problem, makes this technique suitable to be improved by future researches. Both particle swarm optimization and genetic algorithms demonstrated a worse performance in solving the BAP at hand.

The sensitivity analysis on the considered experimental factors, number of machines, type of line, coefficient of variation and minimum throughput rate, confirmed the expected outcomes. On the other hand, the influence of balanced/unbalanced line configurations on the response variable represents a leading finding of the ANOVA analysis. Furthermore, the analysis of the interaction between the type of line balancing and the coefficient of variation revealed another characteristic to be taken into account when the buffer capacity of a flow line has to be designed.

Future research may involve the implementation of new hybrid metaheuristics and the investigation of longer flow lines as well. The proposed PTS could be extended to consider non-linear flow systems including split and merge stations by modifying the generation of the initial solutions and adopting different neighborhood strategies. The segmentation of the system in smaller systems could be another approach [58] to couple with the proposed PTS. Finally, the effect of different types of data distributions on the performance of the metaheuristics should be surveyed.

Appendix A

Procedure: Total buffer capacity and throughput evaluation

```
Step 1: sequencing parts on the first machine
$x_{1,1}=0$
$y_{1,1}=x_{1,1}+t_{1,1}$
For $j=2$ To $J$
  $x_{1,j}=y_{1,j-1}$
  $y_{1,j}=x_{1,j}+t_{1,j}$
Next $j$
Step 2: sequencing parts on the other machines
For $i=2$ To $N$
  For $j=1$ To $J$
    If $j<J$ Then
      If $j=1$ Then
        If $(i-B_j-1)\leq 0$ Then
          $x_{i,j}=y_{i-1,j}$
        Else
          If $y_{i-1,j}>y_{i-Bj-1,j+1}$ Then
            $x_{i,j}=y_{i-1,j}$
          Else
```

```
                $x_{i,j}=y_{i-Bj-1,j+1}$
            End If
        End If
    End If
    If $j\geq 2$ Then
        If $i-B_j-1\leq 0$ Then
            If $y_{i,j-1}\geq y_{i-1,j}$ Then
                $x_{i,j}=y_{i,j-1}$
            Else
                $x_{i,j}=y_{i-1,j}$
            End If
        Else
            If $y_{i,j-1}\geq y_{i-1,j}$ And $y_{i,j-1}\geq y_{i-Bj-1,j+1}$ Then
                $x_{i,j}=y_{i,j-1}$
            ElseIf $y_{i-1,j}\geq y_{i,j-1}$ And $y_{i-1,j}\geq y_{i-Bj-1,j+1}$ Then
                $x_{i,j}=y_{i-1,j}$
            ElseIf $y_{i-Bj-1,j+1}\geq y_{i,j-1}$ And $y_{i-Bj-1,j+1}\geq y_{i-1,j}$ Then
                $x_{i,j}=y_{i-Bj-1,j+1}$
            End If
        End If
    End If
Else
    If $y_{i,j-1}\geq y_{i-1,j}$ Then
        $x_{i,j}=y_{i,j-1}$
    Else
        $x_{i,j}=y_{i-1,j}$
    End If
End If
$y_{i,j}=x_{i,j}+t_{i,j}$
Next $j$
Next $i$
makespan computation: M=$y$(N, J)
throughput computation:$\theta$=N/M
total buffer capacity $Z$=Sum$_k$(Bk)
```

Appendix B

Differential evolution pseudo-code

Step 1: Initialization.
Scale factor *F*, crossover rate *CR*, population size *M* and the maximum number of iterations *S* are initialized. In addition, the whole set of candidate solutions is generated through the ISG procedure. Every variable is defined in the range $[L_k,U_k]$ with $k=1,\ldots,J-1$, where $L_k=1$ and $U_k=N-1$ are lower bound and upper bound of each variable ξ_k, respectively.
Step 2: Mutation.
Any trial solution is mutated according to a target solution. Usually, several equations may be used to handle such target solution (see for example [54,44]). In our implementation of DE the most popular equation has been employed:

$$v_m^{s+1}=\xi_{m3}^s+F(\xi_{m1}^s-\xi_{m2}^s) \tag{13}$$

where *F* is the so-called scale factor, *s* indicates the current iteration and *m*1, *m*2, *m*3 are different integers that are randomly drawn from the set $\{1,\ldots,M\}$.
Step 3: Crossover.
This operator aims to mix variables of a parent solution ξ_k^s and a trial solution v_k^s through the following equation:

$$u_{m,j}^{s+1}=\begin{cases} v_{m,k}^{s+1} & \text{if } rand()<CR \text{ or } k=r_{[1,(K-1)]} \\ \xi_{m,k}^s & \text{otherwise}\end{cases} \tag{14}$$

where *rand*() is a random number in the range [0,1] and $r_{[1,(K-1)]}$ is a random number in the range $[1, K-1]$.
Step 4: Selection.
If the offspring improves the performance of the related parent then it will replace the parent itself on the new population, otherwise that parent will take place in the new population:

$$\xi_m^{s+1}=\begin{cases} u_m^{s+1} & \text{if } f(u_m^{s+1})<f(\xi_m^s) \\ \xi_m^s & \text{otherwise}\end{cases} \tag{15}$$

Step 5: Stopping criterion.
If the maximum number of iteration is reached then the algorithm is stopped; else Steps 3 and 4 are repeated.

Appendix C

Particle swarm optimization pseudo-code

Step 1: *Initialization.*
A swarm of M of particles obtained through the ISG procedure is generated. The position vector of each particle is $\zeta_i(t)=(\zeta_{i,1}(t),\zeta_{i,2}(t),\ldots,\zeta_{i,J-1}(t))$ and its velocity is denoted as $v_i(t)$, where *t* is the index of the current generation.
Step 2: *Velocity computation.*
At each iteration, the particle velocity is computed as follows:

$$v_i(t)=w\times v_i(t-1)+c_i\times r(pbest_i+gbest-2\zeta_i(t)) \tag{16}$$

Where, $pbest_i$is best encountered position of the *i*-th particle, *gbest*is index of the best particle in the swarm, r is a random number in U[0,1], while w_i and c_i are the inertia weight and the acceleration coefficient, respectively.

$$w_i=\left(\lambda-\frac{gbest_i}{pbest_i}\right) \tag{17}$$

$$c_i=\left(\mu+\frac{gbest_i}{pbest_i}\right) \tag{18}$$

Step 3: *Position computation.*
At each iteration, the new particle position depends on the previous position and on the current velocity value, as follows:

$$\zeta_i(t)=\zeta_i(t-1)+v_i(t) \tag{19}$$

References

[1] Alon G, Kroese DP, Raviv T, Rubinstein RY. Application of the Cross-Entropy method to the buffer allocation problem in a simulation-based environment; 2005.
[2] Arumugam MS, Rao MVC, Chandramohan A. A new and improved version of particle swarm optimization algorithm with global-local best parameters. J Knowl Inform Syst (KAIS) 2008;16(3):324–50.
[3] Blum C, Roli A. Metaheuristics optimization: overview and conceptual comparison. ACM Comput Surv 2003;35(3):268–309.
[4] Buzzacot JA, Shantikumar JC. Stochastic models of manufacturing systems. New Jersey: Prentice-Hall; 1993.
[5] Chow W-M. Buffer capacity analysis for sequential production lines with variable processing times. Int J Prod Res 1987;25(8):1183–96.
[6] Coello Coello CA, Mezura Montes E. Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. Adv Eng Inform 2002;16(3):193–203.
[7] Corder GW, Foreman DI. Nonparametric statistics: a step-by-step approach. Wiley; 978-1118840313.
[8] Cruz FRB, Duarte AR, Van Woensel T. Buffer allocation in general single-server queueing networks. Comput Oper Res 2008;35:3581–98.
[9] Dallery Y, Gershwin SB. Manufacturing flow line systems: a review of models and analytical results. Queuing Syst 1992;12(1-2):3–94.
[10] Das S, Suganthan PN. Differential evolution: a survey of the state-of-the-art. IEEE Trans Evol Comput 2011;15(1):27–54.

[11] Deep K, Singh KP, Kansal ML, Mohan C. A real coded genetic algorithm for solving integer and mixed integer optimization problems. Appl Math Comput 2009;212(2):505–18.
[12] Deep K, Thakur M. A new crossover operator for real coded genetic algorithms. Appl Math Comput 2007;188(1):895–911.
[13] Demir L, Tunali S, Eliiyi DT. An adaptive tabu search approach for buffer allocation problem in unreliable non-homogeneous production lines. Comput Oper Res 2012;39:1477–86.
[14] Demir L, Tunali S, Eliiyi DT, Lokketangen A. Two approaches for solving buffer allocation problem in unreliable production lines. Comput Oper Res 2013;40:2556–63.
[15] Demir L, Tunali S, Eliiyi DT. The state of the art on buffer allocation problem: a comprehensive survey. J Intell Manuf 2014;25(3):371–92.
[16] Diamantidis AC, Papadopoulos CT. A dynamic programming algorithm for the buffer allocation problem in homogenous asymptotically reliable serial production lines. Math Prob Eng 2004;3:209–23.
[17] Dolgoui A, Emerev A, Kolokolov A, Sigaev V. A genetic algorithm for the buffer allocation of buffer storage capacities in a production line with unreliable machines. J Math Model Algorithms 2002;1:89–104.
[18] Dolgoui A, Emerev A, Sigaev V. HBBA: hybrid algorithm for buffer allocation in tandem production lines. J Intell Manuf 2007;18:411–20.
[19] Gershwin SB. Manufacturing systems engineering. PTR Prentice Hall; 1994.
[20] Gershwin SB, Schor JE. Efficient algorithms for buffer space allocation. Ann Oper Res 2000;93:117–44.
[21] Glover F, Laguna M. Tabu search. Dordrecth: Kluwer Academic Publishers; 1997.
[22] Hasama M, Ito T, Matsuno S. Optimization of buffer-size allocation using dynamic programming. Int J Math Models Methods Appl Sci 2011;5 (1):125–32.
[23] Hillier FS, So KC. The effect of the coefficient of variation of operation times on the allocation of storage space in production line systems. IIE Trans 1991;23 (2):198–206.
[24] Hillier FS, So KC. The effect of machine breakdowns and interstage storage on the performance of production line systems. Int J Prod Res 1991;29 (10):2043–55.
[25] Huang MG, Chang PL, Chou YC. Buffer allocation in flow-shop-type production systems with general arrival and service patterns. Comput Oper Res 2002;39:103–21.
[26] Jensen PA, Pakat R, Wilson JR. Optimal buffer inventories for multi-stage production systems with failures. Eur J Oper Res 1991;51(3):313–26.
[27] Jeong K-C, Kim Y-D. Algorithms for determining capacities of individual buffers in assembly/disassembly systems. Comput Ind Eng 1997;33 (3−4):605–8.
[28] Kennedy J, Heberhart RC. Particle swarm optimization. Proceeding of the 1995 IEEE international conference on neural network, vol. 4. IEEE Press; 1995. p. 1942.
[29] Kim S, Lee H-J. Allocation of buffer capacity to minimize average work-in-process. Prod Plan Control 2001;12(7):706–16.
[30] Lee H-T, Chen S-K, Chang S. A meta-heuristic approach to buffer allocation on production line. J CCIT 2009;38(1):167–77.
[31] Lutz CM, Davis KR, Sun M. Determining buffer location and size in production lines using a tabu search. Eur J Oper Res 1998;106:301–16.
[32] Mac Gregor Smith J, Cruz FRB. The buffer allocation for general finite buffer queuing networks. IEE Trans 2005;37(4):343–65.
[33] Mann HB, Whitney DR. On a test of whether one of two random variables is stochastically larger than the other. Ann Math Stat 1947;18:50–60.
[34] Massim Y, Yalaoui F, Amodeo L, Zeblah A. Efficient combined immune-decomposition algorithm for optimal buffer allocation in production lines for throughput and profit maximization. Comput Oper Res 2010;37:611–20.
[35] Matta A. Simulation optimization with mathematical programming representation of discrete event systems. December. In: Mason SJ, Hill RR, Moench L, Rose O, Jefferson T, Fowler JF, editors. Proceedings of the 2008 Winter Simulation Conference. Piscataway, New Jersey: Institute of Electrical and Electronic Engineering Inc; 2008. p. 1393–400.
[36] Michalewicz S. Genetic algorithms+data structures=evolution programs. Berlin-Heidelberg: Springer-Verlag; 1994.
[37] Montgomery DG. Design and analysis of experiments. 8 edition. New York: Wiley; 2012.
[38] Nahas N, Ait-Kadi D, Nourelfath M. A new approach for buffer allocation in unreliable production lines; 2006.
[39] Nowicki E, Smutnicki C. The flow shop with parallel machines: a tabu search approach. Eur J Oper Res 1998;106(2−3):226–53.
[40] Papadopoulos CT, O'Kelly MEJ, Vidalis MJ, Spinellis D. Analysis and design of discrete part production lines. New York: Springer Science+Business Media; 2009.
[41] Papadopoulos CT, O'Kelly, Tsadiras AK. A DSS for the buffer allocation of production lines based on a comparative evaluation of a set of search algorithms. Int J Prod Res 2013;51(14):4175–99.
[42] Papadopoulos HT, Heavey C. Queuing theory in manufacturing systems analysis and design: a classification of models for production and transfer lines. Eur J Oper Res 1996;92:1–27.
[43] Papadopoulos HT, Vidalis MI. A heuristic algorithm for the buffer allocation in reliable unbalanced production lines. Comput Ind Eng 2001;41:261–77.
[44] Price KV, Storn RM, Lampinen JA. Differential evolution, a practical approach to global optimization. New York: Springer; 2005.
[45] Reeves CR. Modern heuristic technique. In: Rayward-Smith VJ, Osman IH, Reeves CR, Smith GD, editors. Modern Heuristic Search Methods. 1st ed.. England: John Wiley & sons; 1996.
[46] Richardson JT, Palmer MR, Liepins G, Hilliard M. Some guidelines for genetic algorithms with penalty functions. In: Schaffer D, editor. Proceedings of the Third International Conference on Genetic Algorithms. George Mason University: Morgan Kaufmann Publishers; 1989. p. 191–7.
[47] Sabuncuoglu I, Erel E, Gocgun Y. Analysis of serial production lines: characterization study and a new heuristic procedure for optimal buffer allocation. Int J Prod Res 2006;44(13):2499–523.
[48] Seong D, Chang SY, Hong Y. Heuristic algorithms for buffer allocation in a production line with unreliable machines. Int J Prod Res 1995;33 (7):1989–2005.
[49] Shi L, Shuli M. Optimal buffer allocation in production lines. IEE Trans 2003;35 (1):1–10.
[50] Smith Alice E, Coit David W. Penalty functions handbook of evolutionary computation. Section C 5.2. Oxford University Press and Institute of Physics Publishing; 1996.
[51] Sörensen K, Janssens JK. A Petri net model of a continuous flow transfer line with unreliable machines. Eur J Oper Res 2004;152:248–62.
[52] Spinellis DD, Papadopoulos CT. Stochastic algorithms for buffer allocation in reliable production lines. Math Prob Eng 2000;5:441–58.
[53] Staley D, R, Kim DS. Experimental results for the allocation of buffers in closed serial production lines. Int J Prod Econ 2012;137(2):284–91.
[54] Storn R, Price KV. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. J Global Opt 1997;11(4):341–59.
[55] Storn R, Price K. Differential evolution: a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012. Berkeley, USA: International Computer Science Institute; 1995.
[56] Talbi E-G. Metaheuristics: from design to implementation. Wiley; 0-470-27858-7.
[57] Tsadiras AK, Papadopoulos CT, O'Kelly MEJ. An artificial neural network based decision support system for solving the buffer allocation problem in reliable production lines. Comput Ind Eng 2013;66:1150–62.
[58] Shi C, Gershwin SB. A segmentation approach for solving buffer allocation problems in large production systems. Int J Prod Res 2015 in press.
[59] Yeniay Ö. Penalty function methods for constrained optimization with genetic algorithms. Math Comput Appl 2005;10(1):45–56.