Adaptive Dynamic Jumping Particle Swarm Optimization for Buffer Allocation in Unreliable Production Lines

Kassoul, Khelil; Cheikhrouhou, Naoufel; Zufferey, Nicolas

How to cite

KASSOUL, Khelil, CHEIKHROUHOU, Naoufel, ZUFFEREY, Nicolas. Adaptive Dynamic Jumping Particle Swarm Optimization for Buffer Allocation in Unreliable Production Lines. In: IEEE Access, 2023, vol. 11, p. 90410–90420. doi: 10.1109/ACCESS.2023.3307017

This publication URL: https://archive-ouverte.unige.ch/unige:171810
Publication DOI: 10.1109/ACCESS.2023.3307017

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 12 July 2023, accepted 8 August 2023, date of publication 21 August 2023, date of current version 29 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3307017

RESEARCH ARTICLE

Adaptive Dynamic Jumping Particle Swarm Optimization for Buffer Allocation in Unreliable Production Lines

KHELIL KASSOUL[1], NAOUFEL CHEIKHROUHOU[1,3], (Senior Member, IEEE), AND NICOLAS ZUFFEREY[2]

[1]Geneva School of Business Administration, University of Applied Sciences Western Switzerland (HES-SO), 1227 Geneva, Switzerland
[2]Geneva School of Economics and Management (GSEM), University of Geneva, 1211 Geneva, Switzerland
[3]IFM Business School, 1205 Geneva, Switzerland

Corresponding author: Khelil Kassoul (khelil.kassoul@hesge.ch)

This work was supported by the Swiss National Science Foundation under Grant 100018_182244.

ABSTRACT Over the past five decades, the buffer allocation problem in production lines has been the topic of continuous interest. This paper proposes an adaptive simulation-optimization approach relying on particle swarm optimization (PSO) to solve the buffer allocation problem for unreliable serial production lines. The objective is to maximize the production rate of the production line. The key idea is to integrate a jumping strategy based on logarithmic and exponential functions into the velocity equation of the PSO algorithm using dynamic parameters to achieve quickly (near-)optimal solutions. To evaluate the effectiveness of the proposed method, extensive numerical experiments are conducted using several configurations of production lines, ranging from 3 to 100 machines. Additionally, benchmark algorithms from the literature are employed for comparison purposes. The results indicate that the proposed adaptive approach outperforms the benchmark algorithms regarding efficiency and solution quality.

INDEX TERMS Buffer allocation, particle swarm optimization, production rate, simulation, unreliable production lines.

I. INTRODUCTION

Manufacturing is critical to the global economy and prosperity [1]. A significant body of literature has extensively studied serial production lines utilized in manufacturing systems. These production lines consist of machines arranged sequentially, with buffers between adjacent machines. Units or items, such as materials, parts, or products, traverse these machines following predetermined sequences. Several studies in the manufacturing field aim to enhance the effectiveness of these production lines, considering various objectives such as production rate or profit. However, the efficiency of the production line can be hindered by random factors and events, including machine failures and repairs, random service times, and consequently events such as starvation and blockage. These factors and events can impede the smooth flow of materials and adversely affect the production line's overall performance. One approach to mitigate the impact of these disruptions is by allocating additional buffer sizes along the production line. This buffer allocation strategy helps increase the line's average production rate (PR) while mitigating the propagation of disruptions. Nevertheless, this solution introduces additional challenges as it may lead to higher work-in-process (WIP) inventories, increased capital investment costs, and more floor space requirements. Therefore, determining the optimal allocation of buffers is a challenging optimization problem.in the design of serial production lines called the buffer allocation problem (BAP) [2], [3]. The BAP varies depending on the line's typology and the solution methodology employed. These variations include primal/dual problems, WIP minimization, and profit maximization. Solving the BAP entails developing algorithms based on evaluative search methods (e.g., Markovian state model, decomposition, simulation) or

The associate editor coordinating the review of this manuscript and approving it for publication was Jesus Felez.

generative search methods (e.g., metaheuristics like particle swarm optimization, simulated annealing and genetic algorithms, search algorithms, and dynamic programming).

The BAP can exhibit variations in its objective function and constraints. The primal problem formulation aims to minimize the total buffer capacity needed for the production line while satisfying constraints related to the desired production rate. Conversely, the dual problem formulation aims to maximize the achievable production rate under a total buffer capacity limitation. In the context of constraint profit maximization, the objective is to maximize the profit while adhering to a predetermined total buffer capacity. On the other hand, the unconstraint profit maximization formulation relaxes the constraint on buffer capacity, allowing for more flexibility in the search for the maximum profit.

From the complexity point of view, the BAP is a well-known combinatorial optimization problem classified as NP-hard [4]. Closed-form mathematical description that relate decision variables (e.g., buffer capacities) to performance measures are challenging to develop. Moreover, the computational complexity of the BAP increases significantly with the problem size. Traditional approaches often struggle with local-optima traps, nonlinearity, high-dimensional data, parameter tuning, uncertainty, multi-objective optimization, dynamic environments, and exploration-exploitation trade-offs. Intelligent optimization algorithms, such as Genetic Algorithm (GA) and particle swarm optimization (PSO), efficiently navigate in complex solution spaces, tackle nonlinearity, adapt to changing conditions, and balance competing objectives. By mitigating these shortcomings, these algorithms enhance researchers' ability to address intricate problems and achieve optimal or near-optimal outcomes across diverse fields.

Therefore, the main contributions of this paper are twofold: investigating the performance of PSO in solving the BAP in unreliable production lines and developing a novel variant, named Adaptive Dynamic Jumping Particle Swarm Optimization (APSO for short). The objective function consists of maximizing the production rate of unreliable serial production lines. APSO integrates a leaping strategy based on both logarithmic and exponential functions, as well as dynamic parameters. To the best of our knowledge, only two studies have proposed PSO approaches for the BAP [7], [8]. Another goal of this work is to develop (near-)optimal solutions for different system sizes and various configurations that may help designers make decisions related to the production line design.

The paper is structured as follows: Section II presents a thorough literature review of solution approaches for the BAP. The mathematical formulation of the problem is provided in Section III. Section IV outlines the optimization method and the associated assumptions. The results of numerical experiments are discussed in Section V. Finally, the paper concludes with discussion of the main results summary and highlights potential avenues for future research.

II. LITERATURE REVIEW

The methods used to address the BAP can be categorized into iterative optimization methods (generative and evaluative methods), integrated methods, and explicit solutions [3]. Explicit solutions involve deriving formulas or rules that describe the allocation of buffers based on given optimization problem parameters. These rules and formulas are obtained through exact analytical methods, analysis of optimal solution characteristics, or approximate performance evaluation techniques [9]. Integrated methods employ performance evaluation formulations based on analytical results or sampling and utilize integer programming and standard linear solvers to solve the optimization problem [10], [11]. The iterative method is the most widely used, requiring both generative and evaluative tools [12]. The generative method generates solutions, while the evaluative method assesses their performances.

Evaluation methods can be divided into two categories: analytical methods and simulations. Analytical methods, such as Markovian state models, comprehensively characterize system features but are only suitable for small production lines due to their ample state space and computational complexity [13], [14]. Simulation methods, on the other hand, use discrete event simulation packages such as Arena to evaluate the performance of large and complex production systems [15], [16]. Simulation is beneficial for complex design problems, for which the other assessment techniques may lack precision [2]. However, simulations can be time-consuming as they use multiple replications to provide reliable solutions [17].

There are three commonly used approximation methods for large production lines: the generalized expansion method [18], [19], the decomposition method [20], [21], [22], and the aggregation method [23], [24]. The generalized expansion method utilizes queuing models and split and merge configurations, accommodating reliable machines and random distributed service times. Decomposition methods divide the system into smaller subsystems to reduce computational effort. These methods are applied assuming that service times (resp. repair/failure rates) are either exponentially distributed or deterministic (resp. exponentially or geometrically distributed). Aggregation methods, on the other hand, replace two-machine-and-one-buffer components with equivalent single machines, recursively applying this aggregation until the first or last station is reached.

In terms of generative (optimization) methods, extensive research has focused on finding (near-)optimal values for decision variables [25]. Complete Enumeration (CE) is the simplest generative method but is only suitable for small systems. However, due to computational constraints, no approach efficiently solves the BAP for large production lines [26], [27]. To address the combinatorial nature of the BAP and improve computational efficiency, various optimization methods have been proposed, including dynamic programming, search methods, and metaheuristics.

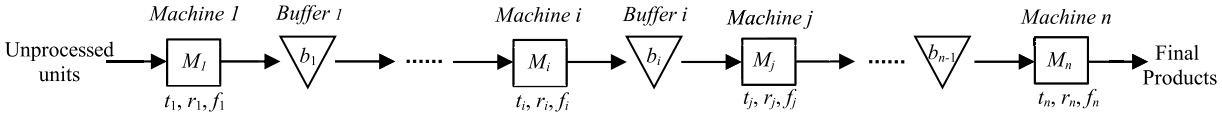

FIGURE 1. Serial production line.

Dynamic programming is a powerful optimization method that divides the BAP into subproblems to reduce the combinatorial complexity [28], [29]. However, it is typically adapted to reliable production lines. Search methods aim to find feasible solutions close to (near-)optimal solutions to overcome the explosion of the number of feasible solutions [2]. Commonly used search methods include the gradient method [30], [31], the degraded ceiling method [32], and Qaraad et al. method [33]. However, search methods are prone to get trapped in local optima and struggle to assess the impact of small changes in buffer capacities on overall system performance. To address these issues, heuristic search algorithms have been developed [34], [35]. Metaheuristics are a category of optimization methods that efficiently manage the search process and explore the solution space within reasonable computing times. Examples of metaheuristics include genetic algorithms (GA) [36], simulated annealing (SA) [37], tabu search (TS) [38], and ant colony optimization [39]. Hybridization of metaheuristics with other methods to enhance their efficiencies has been explored, such as combining TS with Nested Partitions [40], GA with SA [41],PSO and Optimal Computing Budget Allocation (OCBA) [42], and GA with Finite Perturbation analysis (FPA) [43], [44].

Regarding PSO, there are only two works addressing the BAP [7], [8], both focusing on small production lines (8 machines). Lin and Chiu [42] hybridizes PSO with OCBA but for the Resource Allocation Problem. With regards to PSO this paper proposes a new variant of efficient PSO algorithm, namely the APSO wich is well adapted to solve the BAP for both small and large unreliable production lines (from 3 to 100 machines). The efficient convergence of APSO is achieved through integrating a jumping technique in the velocity equation, employing exponential and logarithmic functions. This enables the particles to make leaps and avoid being trapped in local optima. Determining each particle's role and action set is a crucial task, similar to other optimization paradigms involving ''agents'' [45].

III. PROBLEM FORMULATION

Serial production lines, also referred to as flow lines, tandem lines, transfer lines, and similar terms, typically consist of multiple machines through which materials (parts) flow from an external source into the system. The material starts at the first machine, denoted as M_1, then proceeds to the first buffer, known as b_1. From there, it moves on to the next machine, M_2, and continues this progression until it reaches the final machine, M_n, from where it exits the system. Figure 1 illustrates such a production line, demonstrating the connection between n machines $(M_1, \ldots, M_n)$ and their separation by $(n-1)$ buffers $(b_1, \ldots, b_{n-1})$. Each buffer, b_i, represents the capacity between two adjacent machines, M_i and M_{i+1}. Indeed, the machines can be either operational (UP) or non-operational (DOWN) due to internal failures. Several assumptions guide our work: the first machine is always supplied with parts, and the last machine is never blocked; machines can only fail when they are UP and not blocked or starved; the repair and failure rates of machines follow a geometric distribution; delays in part supply are negligible; when a machine is UP, it can be starved (or have a Null Input (NI)) if its upstream buffer is empty, and it can be blocked (or have a Full Output (FO)) if its downstream buffer is full.

To quantify machine reliability, we define the repair probability (r_i) as the likelihood of a machine being UP for the next part, given that it was DOWN for the current part. The failure probability (f_i) represents the probability of a machine being DOWN for the next part, given that it was UP for the current part. The Mean Time Between Failures $(MTBF_i)$ for machine i, with a service time of t_i, is defined as $MTBF_i = t_i/f_i$, while the Mean Time To Repair $(MTTR_i)$ is defined as $MTTR_i = t_i/r_i$.

The total buffer capacity available for allocation across the buffers $b_1, b_2, \ldots, b_{n-1}$ in the serial production line is denoted as B_{max}. The objective of the BAP is to determine the vector $b = (b_1, b_2, \ldots, b_{n-1})$ that maximizes the average production rate while satisfying the constraint $\sum_{i=1}^{n-1} b_i = B_{max}$. The mathematical formulation of the BAP can be represented as follows:

$$\text{Find} \quad b = (b_1, b_2, \ldots, b_{n-1}) \ \textit{so as to maximize} \ f(b) \quad (1)$$

$$\text{Subject to:} \quad \sum_{i=1}^{n-1} b_i = B_{max}; \ b_i \geq 0 \ \textit{and integer} \quad (2)$$

where $f(b)$ represents the average PR of the production line. Let us assume that the system operates for a duration of time T during an experiment, and during this time, the system produces a total of L parts. In this context, we can express the PR of the production line as follows:

$$PR = L/T \quad (3)$$

The complexity of the BAP arises from the absence of an algebraic relationship between line throughput, buffer sizes, and the inherent combinatorial nature of buffer design. While an exact method such as the complete enumeration exists, it becomes impractical due to the exponential increase in feasible allocations as buffer slots (B_{max}) and buffer locations $(n-1)$ grow. Solution methods (e.g., heuristics, metaheuristics) are more suitable for tackling large, realistic instances of the problem. In our research, we employ an adaptive

simulation-optimization approach that relies on the PSO algorithm. The subsequent section delves into the intricacies of this solution approach.

IV. SOLUTION METHOD

A. PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO) is a metaheuristic inspired from the collective behavior of birds or fish flocks, where each particle represents a potential solution and learns from its own experience (personal best position) and the experiences of other particles (global best position) in the search space [46]. The objective is to find the (near-) optimal solution through iterative movements. Choosing PSO can be justified based on several factors. First, PSO stands out due to its simplicity and ease of implementation. It has a straightforward concept and requires minimal parameter tuning, making it accessible to researchers and practitioners. This simplicity facilitates its integration into various applications without extensive computational overhead or intricate implementation procedures. Second, PSO exhibits a desirable convergence speed, often enabling it to reach promising solution-space regions rapidly. This characteristic is advantageous when computing time is critical and prompt optimization results are required. Moreover, efforts to enhance PSO's performance by developing strategies that mitigate premature convergence issues will ensure a more effective search space exploration. Furthermore, PSO inherently balances exploration and exploitation by utilizing personal best and global best information. By leveraging these components with the introduction of novel concepts, PSO efficiently explores the search space while simultaneously exploiting promising regions. This ability to avoid getting trapped in local optima contributes to its effectiveness as an optimization algorithm.

In PSO, each particle is characterized by velocity and position. In the standard PSO algorithm, during each iteration or generation (denoted as it), the position (x_i^{it}) and velocity (v_i^{it}) of each particle i are updated according to Equations (4) and (5) below:

$$v_i^{it+1} = w\, v_i^{it} + c_1 R_1 \left(p_{best_i} it - x_i^{it}\right) + c_2 R_2 \left(p_{gbest} it - x_i^{it}\right) \quad (4)$$

$$x_i^{it+1} = v_i^{it+1} + x_i^{it} \quad (5)$$

Here, w represents the inertia weight coefficient, while c_1 and c_2 are acceleration coefficients. R_1 and R_2 denote two random values sampled from the interval [0, 1]. Similarly, in subsequent equations (mentioned in the following subsection), $R_3, R_4, \ldots, R_{10}$ also represent random values from the interval [0, 1].

Equation (4) updates the velocity of the particle by considering three terms: the inertia term (wv_i^{it}), the cognitive term ($c_1 R_1 \left(p_{best_i} it - x_i^{it}\right)$), and the social term ($c_2 R_2 \left(p_{gbest} it - x_i^{it}\right)$). The inertia term allows the particle to retain part of its previous velocity. The cognitive term adjuststhe velocity based on the particle's personal best position, while the social term adjusts the velocity based on the global best position (p_{gbest}) among all particles.

Equation (5) then updates the particle's position by adding the newly computed velocity. These iterative velocity and position updates allow the particles to explore and exploit the search space to converge toward the (near-)optimal solution.

B. PARAMETERS AND GENERATION OF THE INITIAL SWARM OF APSO

The APSO uses the following parameters. Further information about these parameters can be found in [5].

- s_{wt} : social acceleration worst-coefficient
- c_{wt} : cognitive acceleration worst-coefficient
- c_b : cognitive acceleration best-coefficient
- s_b : social acceleration best-coefficient
- l_w : logarithmic weight parameter
- e_w : exponential weight parameter
- w : inertia weight
- N : population size
- d : damping coefficient
- c_1 : cognitive scaling parameter
- c_2 : social scaling parameter
- $maxG$: maximum number of generations (iterations)

The initial swarm consists of m different particles, denoted as $P = (P_1, P_2, \ldots, P_m)$. Each particle $P_i = \left(p_{i,1}, p_{i,2}, \ldots p_{i,j}, \ldots, p_{i,n-1}\right)$ represents a configuration with n-1 buffer capacities, where $p_{i,j}$ represents the j^{th} buffer (position) of the i^{th} particle. The particle's configurations are generated randomly and uniformly using the discrete uniform distribution, allowing for coverage of various regions in the search space. To satisfy the B_{max} constraint, an adjustment procedure is employed that uniformly increases or decreases specific buffer values of the particle. For a more comprehensive understanding of how these initial solutions are generated, please refer to Kassoul et al. [44].

C. PROPOSED APSO

This subsection provides a detailed description of the proposed APSO Algorithm, which serves two primary purposes. The first purpose is to overcome the limitations of classical PSO, such as premature convergence, by incorporating a jumping strategy consisting of a logarithmic part and an exponential part. The second purpose is to achieve faster convergence through velocity control using dynamic parameters. The pseudocode of APSO is presented in Algorithm 1.

The initial step of APSO involves generating the initial population swarm (as explained in Subsection IV-B) and assigning random velocities to the particles using the discrete uniform distribution $U\{v_{min}, v_{max}\}$ for each coordinate. Here, v_{min} and v_{max} represent the maximum and minimum values of the velocity and are set to the lower and upper bounds of the variable search space, named l_b and u_b, respectively.

Algorithm 1 APSO

Step1: **Initialization**
Set the initial values for the parameters $e_w, l_w, c_b, s_b, c_{wt}, s_{wt}, c_1, c_2, d, w, N, MaxG$.
Initialize the velocity v_i $(i = 1, 2, \ldots, N)$ uniformly and randomly.
Initialize the position x_i $(i = 1, 2, \ldots, N)$ uniformly and randomly.
Evaluate the fitness value of each particle (PR) using the simulation model, and set the initial best position values p_{best_i} and worst position values p_{worst_i} equal to the current x_i $(i= 1, 2, \ldots, N)$.
Determine the global worst and best positions p_{gworst} and p_{gbest}.

Step 2: **Update of the velocity and position of particles**
Divide the population into three equal subswarms N_1, N_2 and N_3.
for the first subswarm N_1, **do**
Calculate ξ and ζ using Equations (6) and (7).
Calculate the velocity v_i for each particle using Equation (8).
for the second subswarm N_2, **do**
Calculate the velocity v_i for each particle using Equation (12).
for the third subswarm N_3, **do**
Calculate the velocity v_i for each particle using Equation (13).
Apply the velocity limits using $v_{i,j} = \max\left(v_{i,j}; v_{min}\right)$ and $v_i = \min\left(v_{i,j}; v_{max}\right)$, where $v_{i,j}$ represents the j^{th} velocity coordinate of the i^{th} particle.
Update the position of particles using Equation (5).
Apply the position limits using $x_{i,j} = \max\left(x_{i,j}; l_b\right)$ and $x_i = \min\left(x_{i,j}; u_b\right)$, where $x_{i,j}$ represents the j^{th} position coordinate of the i^{th} particle.
Step 3: **Update of the personal and global worst/ best positions**
Calculate the fitness values (PR) for each particle.
Update the personal and global worst/ best positions of the particles.
Update the dynamic parameter e_w using Equation (11)
Update the inertia weight w using Equation (14)
Step 4: **Convergence procedure**
Check if the termination criterion is satisfied (i.e., verifying if the maximum number of iterations has been reached).

The next step of the algorithm involves decomposing the swarm population into three equal subswarms, namely N_1, N_2 and N_3, to effectively explore a large portion of the search space. The velocity equation of the first subswarm N_1 incorporates exponential ξ and logarithmic ζ components. These components are expressed in Equations (6) and (7) respectively:

$$\xi^{it} = e^{1/(\|p_i^{it} - x_i^{it}\| + \epsilon)} \tag{6}$$

$$\zeta^{it} = log\left(\left\|p_{best_i}it - x_i^{it}\right\| + \epsilon\right) \tag{7}$$

where ϵ is a very small positive value.

The equation also includes various acceleration coefficients $(s_{wt}, c_{wt}, c_b, s_b)$ and random values $(R_{\text{vec}}, R_1, R_2, R_3, R_4,$ and $R_5)$ to influence the velocity update process. The formulation of the velocity, given in Equation (8), is as follows:

$$\begin{aligned} v_i^{it+1} = {} & w\, v_i^{it} + e_w \xi R_{vec} + l_w \zeta R_1 \left(p_{best_i}it - x_i^{it}\right) \\ & + c_b R_2 \left(p_{best_i}it - x_i^{it}\right) + s_b R_3 \left(p_{gbest}it - x_i^{it}\right) \\ & + c_{wt} R_4 \left(p_{worst_i}^{it} - x_i^{it}\right) + s_{wt} R_5 \left(p_{gworst}^{it} - x_i^{it}\right) \end{aligned} \tag{8}$$

The logarithmic and exponential components play a crucial role in allowing particles to jump when their velocity is near zero, indicating a possible entrapment in a local optimum. By escaping such regions, particles can explore the search space more effectively. To ensure the feasibility of the solution, the velocity and position equations of the proposed method are updated as follows:

$$v_i^{it+1} = \begin{cases} v_{max} & if \;\; v_i^{it+1} > v_{max} \\ v_{min} & if \;\; v_i^{it+1} < v_{min} \\ v_i^{it+1} & otherwise \end{cases} \tag{9}$$

$$x_i^{it+1} = \begin{cases} u_b & if \;\; x_i^{it+1} > u_b \\ l_b & if \;\; x_i^{it+1} < l_b \\ x_i^{it+1} & otherwise \end{cases} \tag{10}$$

To strike a balance between exploration and exploitation, the exponential weight parameter e_w is utilized. It decreases linearly using the damping coefficient d, facilitating global exploration with large leaps at the beginning of the search and localized exploitation with small jumps in specific regions. The exponential weight parameter e_w is updated according to Equation (11).

$$e_w^{it+1} = d e_w^{it} \tag{11}$$

The velocity update of the first subswarm considers the global and personal worst positions to maintain population diversity and explore new regions in the solution space. Negative values are assigned to the coefficient parameters (c_{wt} and s_{wt}) associated with the personal and global worst positions, respectively, allowing for investigation of opposite solution-space regions visited by these positions.

For the second subpopulation N_2, the velocity update incorporates the logarithmic and exponential components, previous velocity, and global and personal best positions. The formulation of the velocity is given in Equation (12).

$$\begin{aligned} v_i^{it+1} = {} & w\, v_i^{it} + e_w \xi R_{vec} + l_w \zeta R_6 \left(p_{best_i}it - x_i^{it}\right) \\ & + c_b R_7 \left(p_{best_i}it - x_i^{it}\right) + s_b R_8 \left(p_{gbest}it - x_i^{it}\right) \end{aligned} \tag{12}$$

Lastly, the velocity update of the subswarm N_3 follows the C_NPSO procedure proposed by [5] and differs from the standard PSO by setting $c_1 = -1$ and $c_2 = 2$. The equation for velocity update is provided in Equation (13).

$$\begin{aligned} v_i^{it+1} = {} & w v_i^{it} + c_1 R_9 \left(p_{best_i}it - x_i^{it}\right) \\ & + c_2 R_{10} \left(p_{gbest}it - x_i^{it}\right) \end{aligned} \tag{13}$$

The value of the inertia weight coefficient (w) in APSO decreases linearly from an initial value to a final value. This adjustment gradually reduces the particles' ability to perform a global search while increasing their ability to conduct a local search. w is updated using Equation (14).

$$w^{it+1} = d\left(\frac{1 - w^{it}}{1 + w^{it}}\right) \tag{14}$$

In summary, the APSO Algorithm encompasses a jumping strategy and dynamic parameter control to address the limitations of classical PSO. It divides the swarm into subswarms for efficient exploration across the search area, employing different velocity updates with logarithmic and exponential components. These enable particles to jump when their velocity nears zero, preventing them from being stuck in local optima. Additionally, the algorithm includes various acceleration coefficients, and dynamic parameter within the velocity equation, allowing effective exploration and exploitation of the search space.

V. NUMERICAL EXPERIMENTS

The experiments are conducted on a computer with a 2.4-GHz Core (TM) i5 CPU and 16 GB of RAM. The discrete-event simulation models are developed using the Arena simulation language V14.0, and the algorithms are implemented in Java. The APSO algorithm is terminated after reaching a maximum of 50 generations ($MaxG = 50$).

The study considers three different sizes of serial production lines: small lines with 3, 5, and 10 machines, medium lines with 20 machines, and large lines with 40 and 100 machines. All machines in the production lines are identical and unreliable. The repair and failure times are geometrically distributed. The service time for each machine (t_i) is set to 1. The final buffer allocation, average PR values, and convergence features are analyzed in the study.

To ascertain the effectiveness of the proposed APSO method, we conduct a comparative analysis with seven state-of-the-art algorithms that are well-established for solving the BAP. The selection of these algorithms is rooted in their demonstrated capabilities, and their performance data and outcomes are accessible for a subset of the considered instances. This deliberate selection ensures a meaningful and relevant evaluation of APSO's performance when compared to established benchmarks. The algorithms being compared are GT (Gradient Technique) [47], DGT (Dual Gradient Technique) [30], DC (Degraded Ceiling method and decomposition approximation) [32], IDA (Immune Decomposition Algorithm) [48], ADA-TS (Tabu Search with Analytical Decomposition Approximation) ([49], GA-SA (Simulated Annealing and Genetic Algorithm) [41], and GA-FPA (Genetic Algorithm and Finite Perturbation Analysis) [46]. The buffer allocation and the average PR are reported for each experiment. The results of the best algorithm are highlighted in bold.

Figure 2 illustrates the convergence curves of APSO for representative instances with n values of 5, 10, 20, and 40. We highlight that all simulation runs in this study are performed using identical experimental conditions. Table 1 presents detailed information regarding the number of simulated units or parts, as well as the number of runs (replications) for each example. These specific numbers are carefully selected to ensure equitable comparisons with other methods and to obtain stable results with consistent average PR values.

TABLE 1. Number of simulated units and replications for the six comparative examples.

Instance size	3	5	10	20	40	100
Units	10,000	100,000	10,000	10,000	10,000	100,000
Replications	30	50	30	5	5	10

Although a detailed discussion of computation times is not provided, it is worth noting that the average computation time to obtain the best solution is approximately 96 minutes for a large instance with $n = 40$ machines. This duration, when scrutinized against insights gleaned from diverse industrial investigations within the production field [50], [51], [52], emerges as a judicious timeframe. Importantly, this observation offers more than just a ballpark estimate; it sheds light on the time investment necessary for attaining optimal outcomes for numerous real-world industrial contexts. This delineates the capacity of intelligent optimization techniques, like the APSO approach, to mitigate time complexity challenges by yielding optimal or near-optimal solutions within practical timeframes, even for intricate and expansive production line scenarios.

A. RESULTS ON A SMALL INSTANCE WITH 3 MACHINES

Table 2 presents the repair and failure rates for the instance proposed by Gershwin and Schor [30], who consider 3 machines and where the buffer configuration and production rate (PR) for this instance are not explicitly mentioned. Table 3 provides a comparison between IDA, GA-FPA, and APSO. Interestingly, APSO and GA-FPA yield the same PR, indicating their comparable performance in finding an optimal solution. Moreover, all three methods exhibit a similar buffer allocation scheme, where the first buffer is larger than the second buffer. This observation can be attributed to the relatively higher values of failure rate (f_1) and repair rate (r_1) associated with the first machine compared to the second machine (f_2 and r_2, respectively).

TABLE 2. Parameters of the considered instance.

Machine	1	2	3
r_i	0.35	0.15	0.4
f_i	0.037	0.015	0.02

TABLE 3. Results on an instance with n = 3 machines and $B_{max} = 2$.

Method	Buffer allocation		PR
IDA	14	6	0.86799
GA-FPA	13	7	**0.87178**
APSO	13	7	**0.87178**

The preference for a larger buffer at the beginning of the production line can be explained by the need to accommodate

potential disruptions caused by the first machine's higher failure and repair rates. By allocating a larger buffer, the system can mitigate the impact of downtime and maintain a smoother materials flow.

B. RESULTS ON A SMALL INSTANCE WITH 5 MACHINES

Table 4 provides the repair and failure rates for the instance proposed by Ho et al. [47], who consider 5 machines. Table 5 presents the results of GT, DGT, IDA, ADA-TS, GA-SA, GA-FPA and APSO. Among these methods, APSO achieves the highest PR value, indicating its superior performance in maximizing the PR. APSO reaches this optimal configuration within the first 15 generations, as illustrated in Figure 2.

An interesting observation from the results is that all the approaches, including APSO and the other algorithms, converge to configurations where smaller buffer slots are allocated at the ends of the production line. This allocation strategy is advantageous as it facilitates the smooth passage of parts and reduces the likelihood of line blockages. The system can maintain an uninterrupted production flow by allowing for smaller buffer slots at the ends, thereby improving overall productivity.

TABLE 4. Parameters of the considered instance.

Machine	1	2	3	4	5
MTTR = $1/r_i$	11	19	12	7	7
MTBF = $1/f_i$	20	167	22	22	26

TABLE 5. Results on an instance with n = 5 machines and $B_{max} = 31$.

Method	Buffer allocation	PR
GT	5 11 8 7	0.4914
DGT	7 10 10 4	0.4943
IDA	6 10 11 4	0.4941
ADA-TS	7 10 10 4	0.4943
GA-SA	7 10 10 4	0.4943
GA-FPA	7 11 9 4	0.4948
APSO	5 11 12 3	**0.4965**

C. RESULTS ON A SMALL INSTANCE WITH 10 MACHINES

Table 6 provides the repair and failure rates related explicitly to the instance proposed by Nahas et al. [32], who considers 10 machines. Table 7 presents the comparative results of DC, IDA, ADA-TS, GA-SA, GA-FPA and APSO. By examining the data provided in Table 7, we can observe that APSO achieves the second-best PR value among the algorithms evaluated. Figure 2 highlights that APSO reaches its optimal buffer allocation solution relatively quickly. Specifically, the best buffer allocation for APSO is achieved before completing 25 generations, demonstrating its efficiency in finding the optimal solution quickly.

D. RESULTS ON MEDIUM INSTANCE WITH 20 MACHINES

In Table 8, the first two columns present the repair and failure rates specifically for the instance proposed by Demir et al. [49], who consider lines with 10 machines. In this instance, all machines exhibit similar repair and failure rates, ranging from 0.1 to 0.9, with an increment of 0.1. This implies that 9 cases are considered in this example, each corresponding to a different combination of repair and failure rates.

The subsequent columns in Table 8 provide the PR values obtained by the different methods: ADA-TS, GA-SA, GA-FPA, and APSO. Upon examination, it becomes evident that APSO outperforms both GA-SA and ADA-TS regarding PR values. This indicates that APSO consistently achieves higher-quality solutions compared to the other two methods for this instance. Interestingly, APSO and GA-FPA exhibit similar performances, as both methods reach the best results in four cases out of the nine cases considered. This demonstrates their comparative effectiveness in finding optimal solutions.

Additionally, it is worth noting that APSO attains its best PR values before reaching the generation #30, implying that it converges relatively quickly to better solutions. This observation aligns with expectations, as it is generally anticipated that the PR values would increase with the values of r_i and f_i for all the methods tested.

TABLE 6. Parameters of the considered instance.

Machine	1	2	3	4	5	6	7	8	9	10
MTTR =$1/r_i$	7	7	5	10	9	14	5	8	10	10
MTBF =$1/f_i$	20	30	22	22	25	40	23	30	45	20

TABLE 7. Results on an instance with n = 10 machines and $B_{max} = 270$.

Method	Buffer allocation	PR
DC	14 19 30 54 45 27 23 24 34	0.64135
IDA	14 19 30 52 47 27 23 24 34	0.64139
ADA-TS	14 19 30 54 45 27 23 24 34	0.64135
GA-SA	7 16 48 61 24 41 20 34 19	0.63016
GA-FPA	19 23 24 45 43 34 22 29 31	**0.64920**
APSO	13 24 27 42 44 35 24 30 31	0.64850

E. RESULTS ON LARGE INSTANCES WITH 40 MACHINES

Table 9 follows the same structure as Table 8 and pertains to a larger instance proposed by Demir et al. [49]. The comparison involves the same methods as in Table 8.

It is noteworthy that APSO is notably more efficient than the other methods when applied to this larger instance. In fact, APSO delivers the best solution in 7 out of the 9 cases evaluated. This finding suggests that APSO's efficiency improves as the instance size increases, making it a more effective approach than the alternative methods. Similar to the previous

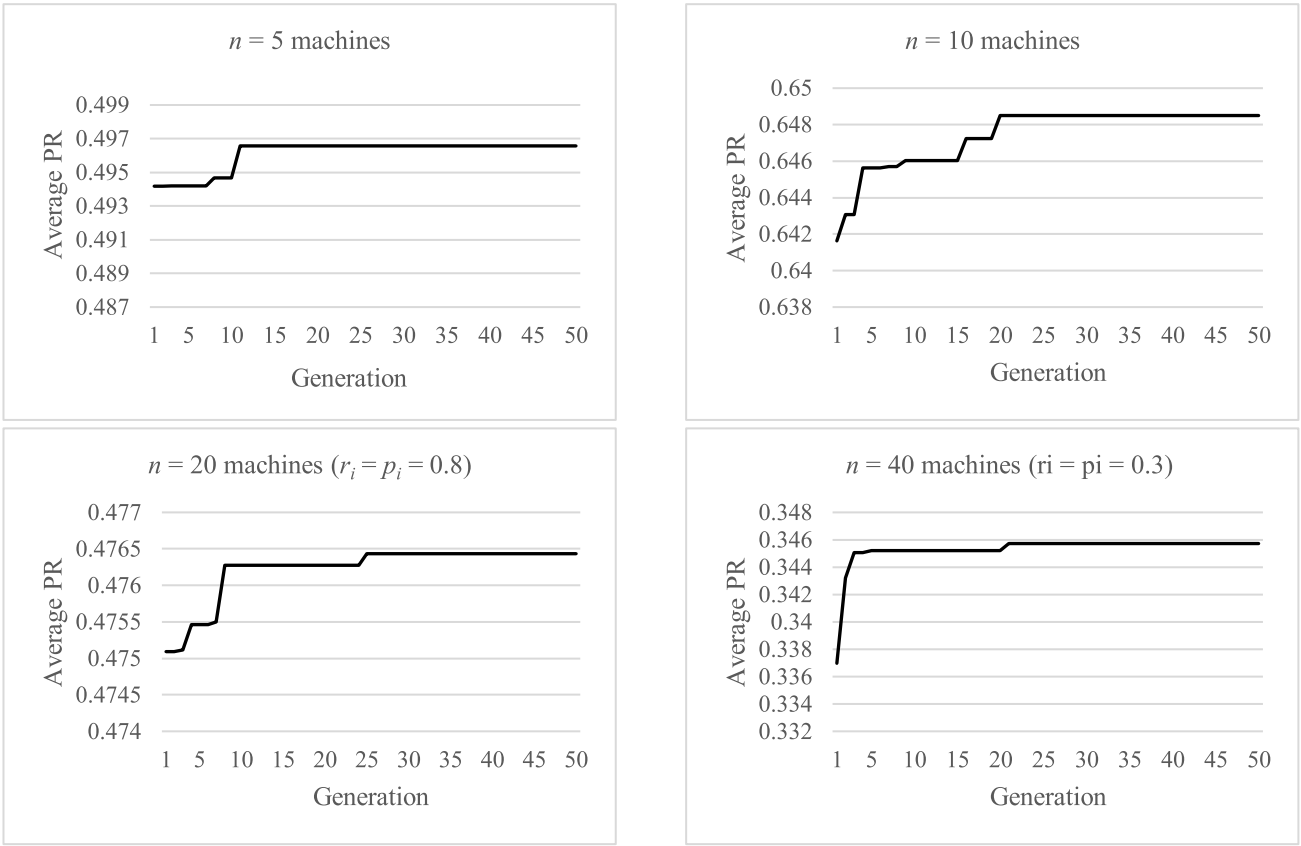

FIGURE 2. Convergence curves of APSO for various instances with n belonging to the set {5, 10, 20, 40}.

TABLE 8. Results on an instance with n = 20 machines and $B_{max} = 100$.

Parameters		Average PR			
r_i	f_i	APSO	GA-FPA	GA-SA	ADA-TS
0.1	0.1	0.221544	0.223559	0.226499	**0.234191**
0.2	0.2	0.304041	**0.305203**	0.302448	0.297025
0.3	0.3	**0.355727**	0.354811	0.349517	0.334450
0.4	0.4	0.393448	**0.394316**	0.354299	0.359637
0.5	0.5	0.421256	**0.422175**	0.382401	0.377895
0.6	0.6	**0.443987**	0.443916	0.397120	0.391846
0.7	0.7	**0.461614**	0.461407	0.446904	0.402760
0.8	0.8	0.476434	**0.476649**	0.450032	0.411638
0.9	0.9	**0.487833**	0.487760	0.459111	0.419018

instance, it is worth noting that PR increases with respect to r_i and f_i.

F. RESULTS ON A LARGE INSTANCE WITH 100 MACHINES

Works on BAPs for very large production lines have been relatively limited, such as those conducted by Kose and Kilincci [41] and Spinellis et al. [18]. Researchers in this field have found that obtaining competitive results for such problems requires extensive computation times, often spanning several hours. The computation time exhibits exponential growth as the number of machines n increases. In this example, we focus on a production line of 100 unreliable machines proposed by Kose and Kilincci [41]. The repair and failure rates for these machines follow a geometric distribution where $r_i = 0.5$ and $f_i = 0.05$.

Due to the demanding nature of the required computations, the experiment is conducted with fewer generations. Expressly, MaxG is set to 20 instead of 50. Table 10 provides a comparative analysis of the results obtained from GA-SA and APSO algorithms. The computation times in seconds for each method are listed in the last column. Upon examination, it becomes evident that APSO outperforms GA-SA significantly regarding both solution quality (considering the PR values) and speed (considering the computing times required to obtain the best solution). This significant performance gap suggests that APSO offers a favorable trade-off between solution quality and computation time.

G. BEHAVIOR OF THE RESULTS WHEN N AUGMENTS

Relying on Figure 2 and Tables 3, 5, 7 to 10, several observations can be made regarding the augmentation of n which represents the number of machines in the system. One important finding is that APSO demonstrates increased competitiveness compared to other methods as n grows larger. APSO tends to achieve (near-)optimal solutions while ensuring a significant margin is maintained within the allowed number

TABLE 9. Results on an instance with n = 40 machines and $B_{max} = 200$.

Parameters		Average PR			
r_i	f_i	APSO	GA-FPA	GA-SA	ADA-TS
0.1	0.1	0.212987	0.214819	0.220132	**0.221273**
0.2	0.2	**0.293629**	0.293308	0.270289	0.282169
0.3	0.3	**0.345723**	0.344024	0.328518	0.325256
0.4	0.4	**0.383742**	0.383456	0.365301	0.351494
0.5	0.5	**0.412375**	0.412104	0.371244	0.370666
0.6	0.6	**0.435811**	0.435792	0.404937	0.385328
0.7	0.7	**0.455184**	0.454676	0.446904	0.397255
0.8	0.8	**0.470614**	0.470529	0.468270	0.406559
0.9	0.9	0.484953	**0.485077**	0.482113	0.413990

TABLE 10. Results on an instance with n = 100 machines and $B_{max} = 300$.

Method	PR	CPU (s)
GA-SA	0.742554	125588.6
APSO	**0.759528**	**96750.1**

of generations (MaxG=50). For instance, in Figure 2, APSO reaches its optimal solution at generation 11 with n equal to 5, at generation 20 with n equal to 10, at generation 25 with n equal to 20, and at generation 22 with n equal to 40. This pattern suggests a high level of stability across different problem sizes, indicating that APSO effectively balances its exploration and exploitation search abilities throughout the search process.

Regarding allocating buffer capacity for production lines of varying sizes, APSO consistently generates the best buffer allocation in 14 out of the 22 cases, showcasing its superior performance. Even in cases where APSO is not ranked first, it still produces highly competitive results. However, the observed patterns of buffer allocation are often unexpected, underscoring the complexity of the problem. Notably, the buffer values in the middle of the line tend to be similar and relatively larger compared to the buffer capacities at the end of the production line. Additionally, as the number of machines increases, the optimal buffer values tend to become identical.

VI. CONCLUSION

This study introduces a new variant of PSO, the Adaptive Dynamic Jumping Particle Swarm Optimization (APSO) for the Buffer Allocation Problem in unreliable production lines. Results of the application of the algorithm to different sizes and configurations of productions lines show that it is efficient and reliable when the production rate is maximized. As APSO incorporates a jumping strategy, utilizing exponential and logarithmic functions and dynamic parameters within the velocity equation, such strategy enables it to explore the solution space effectively and converge more rapidly than state of the art optimization techniques regarding solution quality and computation time, highlighting its superiority in solving the buffer allocation problem. The experimental evaluation encompasses various instance sizes, ranging from small-scale systems with $n = 3$ machines to larger systems with $n = 100$ machines. The results demonstrate that APSO performs consistently well across different problem sizes, showcasing its capability to scale up without reducing its performance. The observed patterns in buffer allocation underscore the challenging nature of the problem, with middle buffers having similar and larger capacities compared to the end buffers and convergence of buffer values as the number of machines increases.

A promising avenue for future research lies in the exploration of hybridizing the APSO approach with other metaheuristic techniques. This innovative fusion could entail the integration of sophisticated learning mechanisms introduced by Thevenin and Zufferey [53] and by Schindl and Zufferey [54], into local-search methodologies. By incorporating these advanced learning strategies, APSO's capabilities could be enhanced, leading to potentially superior convergence and solution quality.

Moreover, a valuable extension of the proposed algorithm could involve its adaptation to address more intricate production line scenarios, such as assembly/disassembly systems. Additionally, a noteworthy future direction involves extending APSO's applicability to encompass multi-objective optimization scenarios. This expansion would not only enhance APSO's versatility but also reinforce its effectiveness in tackling real-world production optimization challenges across diverse operational contexts.

DISCLOSURE STATEMENT

The authors report there are no competing interests to declare.

DATA AVAILABILITY STATEMENT

The paper includes no data.

ACKNOWLEDGMENT

This paper is based on chapter 2 of the Ph.D. Thesis of the first author [55].

REFERENCES

[1] J. Li and S. M. Meerkov, "Mathematical modeling of production systems," in *Production Systems Engineering*. Boston, MA, USA: Springer, 2009, pp. 1–59, doi: 10.1007/978-0-387-75579-3_3.

[2] L. Demir, S. Tunali, and D. T. Eliiyi, "The state of the art on buffer allocation problem: A comprehensive survey," *J. Intell. Manuf.*, vol. 25, no. 3, pp. 371–392, Jun. 2014.

[3] S. Weiss, J. A. Schwarz, and R. Stolletz, "The buffer allocation problem in production lines: Formulations, solution methods, and instances," *IISE Trans.*, vol. 51, no. 5, pp. 456–485, May 2019.

[4] J. Macgregor Smith and F. R. B. Cruz, "The buffer allocation problem for general finite buffer queueing networks," *IIE Trans.*, vol. 37, no. 4, pp. 343–365, Feb. 2005.

[5] K. Kassoul, N. Zufferey, N. Cheikhrouhou, and S. B. Belhaouari, "Exponential particle swarm optimization for global optimization," *IEEE Access*, vol. 10, pp. 78320–78344, 2022, doi: 10.1109/ACCESS.2022.3193396.

[6] K. Kassoul, S. B. Belhaouari, and N. Cheikhrouhou, "Dynamic cognitive-social particle swarm optimization," in *Proc. 7th Int. Conf. Autom., Robot. Appl. (ICARA)*, Feb. 2021, pp. 200–205, doi: 10.1109/ICARA51699.2021.9376550.

[7] K. L. Narasimhamu, V. V. Reddy, and C. S. P. Rao, "Optimal buffer allocation in tandem closed queuing network with single server using PSO," *Proc. Mater. Sci.*, vol. 5, pp. 2084–2089, Jan. 2014, doi: 10.1016/j.mspro.2014.07.543.

[8] K. L. Narasimhamu, V. V. Reddy, and C. S. P. Rao, "Optimization of buffer allocation in manufacturing system using particle swarm optimization," *Int. Rev. Model. Simulations (IREMOS)*, vol. 8, no. 2, p. 212, Apr. 2015, doi: 10.15866/iremos.v8i2.5666.

[9] A. K. Tsadiras, C. T. Papadopoulos, and M. E. J. O'Kelly, "An artificial neural network based decision support system for solving the buffer allocation problem in reliable production lines," *Comput. Ind. Eng.*, vol. 66, no. 4, pp. 1150–1162, Dec. 2013.

[10] A. Alfieri and A. Matta, "Mathematical programming formulations for approximate simulation of multistage production systems," *Eur. J. Oper. Res.*, vol. 219, no. 3, pp. 773–783, Jun. 2012.

[11] G. Pedrielli, A. Alfieri, and A. Matta, "Integrated simulation–optimisation of pull control systems," *Int. J. Prod. Res.*, vol. 53, no. 14, pp. 4317–4336, Jul. 2015.

[12] D. D. Spinellis and C. T. Papadopoulos, "A simulated annealing approach for buffer allocation in reliable production lines," *Ann. Oper. Res.*, vol. 93, no. 1, pp. 373–384, 2000.

[13] D. C. Alexandros and P. T. Chrissoleon, "Exact analysis of a two-workstation one-buffer flow line with parallel unreliable machines," *Eur. J. Oper. Res.*, vol. 197, no. 2, pp. 572–580, Sep. 2009.

[14] A. Dolgui, A. Eremeev, and V. Sigaev, "On local optima distribution in buffer allocation problem for production line with unreliable machines," *IFAC-PapersOnLine*, vol. 55, no. 10, pp. 1092–1097, 2022.

[15] M. Amiri and A. Mohtashami, "Buffer allocation in unreliable production lines based on design of experiments, simulation, and genetic algorithm," *Int. J. Adv. Manuf. Technol.*, vol. 62, nos. 1–4, pp. 371–383, Sep. 2012.

[16] S. Weiss, A. Matta, and R. Stolletz, "Optimization of buffer allocations in flow lines with limited supply," *IISE Trans.*, vol. 50, no. 3, pp. 191–202, Mar. 2018.

[17] S. Helber, K. Schimmelpfeng, R. Stolletz, and S. Lagershausen, "Using linear programming to analyze and optimize stochastic flow lines," *Ann. Operations Res.*, vol. 182, no. 1, pp. 193–211, Jan. 2011.

[18] D. Spinellis, C. Papadopoulos, and J. M. Smith, "Large production line optimization using simulated annealing," *Int. J. Prod. Res.*, vol. 38, no. 3, pp. 509–541, Feb. 2000.

[19] H.-Y. Zhang, Q.-X. Chen, J. M. Smith, N. Mao, A.-L. Yu, and Z.-T. Li, "Performance analysis of open general queuing networks with blocking and feedback," *Int. J. Prod. Res.*, vol. 55, no. 19, pp. 5760–5781, Oct. 2017.

[20] A. Diamantidis, J.-H. Lee, C. T. Papadopoulos, J. Li, and C. Heavey, "Performance evaluation of flow lines with non-identical and unreliable parallel machines and finite buffers," *Int. J. Prod. Res.*, vol. 58, no. 13, pp. 3881–3904, Jul. 2020.

[21] G. Liberopoulos, "Performance evaluation of a production line operated under an echelon buffer policy," *IISE Trans.*, vol. 50, no. 3, pp. 161–177, Mar. 2018.

[22] S. Xi, Q. Chen, J. MacGregor Smith, N. Mao, A. Yu, and H. Zhang, "A new method for solving buffer allocation problem in large unbalanced production lines," *Int. J. Prod. Res.*, vol. 58, no. 22, pp. 6846–6867, Nov. 2020.

[23] Y. Bai, J. Tu, M. Yang, L. Zhang, and P. Denno, "A new aggregation algorithm for performance metric calculation in serial production lines with exponential machines: Design, accuracy and robustness," *Int. J. Prod. Res.*, vol. 59, no. 13, pp. 4072–4089, Jul. 2021.

[24] O. Abdelrahman and P. Keikhosrokiani, "Assembly line anomaly detection and root cause analysis using machine learning," *IEEE Access*, vol. 8, pp. 189661–189672, 2020, doi: 10.1109/ACCESS.2020.3029826.

[25] C. T. Papadopoulos, M. E. J. O'Kelly, and A. K. Tsadiras, "A DSS for the buffer allocation of production lines based on a comparative evaluation of a set of search algorithms," *Int. J. Prod. Res.*, vol. 51, no. 14, pp. 4175–4199, Jul. 2013.

[26] F. S. Hillier and K. C. So, "The effect of the coefficient of variation of operation times on the allocation of storage space in production line systems," *IIE Trans.*, vol. 23, no. 2, pp. 198–206, Jun. 1991.

[27] W. Zhou and Z. Lian, "A tandem network with a sharing buffer," *Appl. Math. Model.*, vol. 35, no. 9, pp. 4507–4515, Sep. 2011.

[28] A. C. Diamantidis and C. T. Papadopoulos, "A dynamic programming algorithm for the buffer allocation problem in homogeneous asymptotically reliable serial production lines," *Math. Problems Eng.*, vol. 2004, no. 3, pp. 209–223, 2004.

[29] T. Alfakih, M. M. Hassan, and M. Al-Razgan, "Multi-objective accelerated particle swarm optimization with dynamic programing technique for resource allocation in mobile edge computing," *IEEE Access*, vol. 9, pp. 167503–167520, 2021, doi: 10.1109/ACCESS.2021.3134941.

[30] S. B. Gershwin and J. E. Schor, "Efficient algorithms for buffer space allocation," *Ann. Oper. Res.*, vol. 93, no. 1, pp. 117–144, 2000.

[31] C. Shi and S. B. Gershwin, "A segmentation approach for solving buffer allocation problems in large production systems," *Int. J. Prod. Res.*, vol. 54, no. 20, pp. 6121–6141, Oct. 2016.

[32] N. Nahas, D. Ait-Kadi, and M. Nourelfath, "A new approach for buffer allocation in unreliable production lines," *Int. J. Prod. Econ.*, vol. 103, no. 2, pp. 873–881, Oct. 2006.

[33] M. Qaraad, S. Amjad, N. K. Hussein, S. Mirjalili, N. B. Halima, and M. A. Elhosseini, "Comparing SSALEO as a scalable large scale global optimization algorithm to high-performance algorithms for real-world constrained optimization benchmark," *IEEE Access*, vol. 10, pp. 95658–95700, 2022, doi: 10.1109/ACCESS.2022.3202894.

[34] I. Sabuncuoglu, E. Erel, and Y. Gocgun, "Analysis of serial production lines: Characterisation study and a new heuristic procedure for optimal buffer allocation," *Int. J. Prod. Res.*, vol. 44, no. 13, pp. 2499–2523, Jul. 2006.

[35] L. Demir, A. C. Diamantidis, D. T. Eliiyi, M. E. O'Kelly, and S. Tunali, "Optimal buffer allocation for serial production lines using heuristic search algorithms: A comparative study," *Int. J. Ind. Syst. Eng.*, vol. 33, no. 2, pp. 252–270, 2019.

[36] Y. Alaouchiche, Y. Ouazene, and F. Yalaoui, "Multi-objective optimization of energy-efficient buffer allocation problem for non-homogeneous unreliable production lines," *IEEE Access*, vol. 10, pp. 3320–3335, 2022, doi: 10.1109/ACCESS.2021.3139954.

[37] X. Han, Y. Dong, L. Yue, and Q. Xu, "State transition simulated annealing algorithm for discrete-continuous optimization problems," *IEEE Access*, vol. 7, pp. 44391–44403, 2019, doi: 10.1109/ACCESS.2019.2908961.

[38] S. Gao, "A bottleneck detection-based Tabu search algorithm for the buffer allocation problem in manufacturing systems," *IEEE Access*, vol. 10, pp. 60507–60520, 2022, doi: 10.1109/ACCESS.2022.3181134.

[39] H. Peng, C. Ying, S. Tan, B. Hu, and Z. Sun, "An improved feature selection algorithm based on ant colony optimization," *IEEE Access*, vol. 6, pp. 69203–69209, 2018, doi: 10.1109/ACCESS.2018.2879583.

[40] L. Shi and S. Men, "Optimal buffer allocation in production lines," *IIE Trans.*, vol. 35, no. 1, pp. 1–10, Jan. 2003.

[41] S. Y. Kose and O. Kilincci, "Hybrid approach for buffer allocation in open serial production lines," *Comput. Operations Res.*, vol. 60, pp. 67–78, Aug. 2015.

[42] J. T. Lin and C.-C. Chiu, "A hybrid particle swarm optimization with local search for stochastic resource allocation problem," *J. Intell. Manuf.*, vol. 29, no. 3, pp. 481–495, Mar. 2018.

[43] K. Kassoul, N. Cheikhrouhou, and N. Zufferey, "Simultaneous allocation of buffer capacities and service times in unreliable production lines," *Int. J. Prod. Res.*, pp. 1–21, Jan. 2023, doi: 10.1080/00207543.2023.2168310.

[44] K. Kassoul, N. Cheikhrouhou, and N. Zufferey, "Buffer allocation design for unreliable production lines using genetic algorithm and finite perturbation analysis," *Int. J. Prod. Res.*, vol. 60, no. 10, pp. 3001–3017, May 2022, doi: 10.1080/00207543.2021.1909169.

[45] N. Zufferey, "Optimization by ant algorithms: Possible roles for an individual ant," *Optim. Lett.*, vol. 6, no. 5, pp. 963–973, Jun. 2012.

[46] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.

[47] Y. C. Ho, M. A. Eyler, and T. T. Chien, "A gradient technique for general buffer storage design in a production line," *Int. J. Prod. Res.*, vol. 17, no. 6, pp. 557–580, Nov. 1979.

[48] Y. Massim, F. Yalaoui, L. Amodeo, E. Chatelet, and A. Zeblah, "Efficient combined immune-decomposition algorithm for optimal buffer allocation in production lines for throughput and profit maximization," *Comput. Oper. Res.*, vol. 37, no. 4, pp. 611–620, Apr. 2010.

[49] L. Demir, S. Tunali, and A. Løkketangen, "A Tabu search approach for buffer allocation in production lines with unreliable machines," *Eng. Optim.*, vol. 43, no. 2, pp. 213–231, Feb. 2011.

[50] J. Respen, N. Zufferey, and E. Amaldi, "Metaheuristics for a job scheduling problem with smoothing costs relevant for the car industry," *Networks*, vol. 67, no. 3, pp. 246–261, May 2016.

[51] N. Zufferey, "Tabu search approaches for two car sequencing problems with smoothing constraints," in *Metaheuristics for Production Systems*. Cham, Switzerland: Springer, 2016, pp. 167–190.
[52] J. Respen, N. Zufferey, and P. Wieser, "Three-level inventory deployment for a luxury watch company facing various perturbations," *J. Oper. Res. Soc.*, vol. 68, no. 10, pp. 1195–1210, Oct. 2017.
[53] S. Thevenin and N. Zufferey, "Learning variable neighborhood search for a scheduling problem with time windows and rejections," *Discrete Appl. Math.*, vol. 261, pp. 344–353, May 2019.
[54] D. Schindl and N. Zufferey, "A learning Tabu search for a truck allocation problem with linear and nonlinear cost components," *Nav. Res. Logistics (NRL)*, vol. 62, no. 1, pp. 32–45, Feb. 2015, doi: 10.1002/nav.21612.
[55] K. Kassoul, "Hybrid algorithms for designing production lines," Ph.D. dissertation, Geneva School Econ. Manag. (GSEM), Univ. Geneva, Geneva, Switzerland, 2022.

KHELIL KASSOUL received the Ph.D. degree in economics and management from the Geneva School of Economics and Management (GSEM). He was an Associate Member of the European Organization for Nuclear Research (CERN). From 2003 to 2021, he held the position of a Teacher of mathematics and physics for Swiss Maturity and Baccalaureate classes. He is currently a Researcher with the Geneva School of Business Administration (HEG)–University of Applied Sciences of Western Switzerland (HES-SO). His research interests include applied mathematics, production systems modeling, simulation and optimization, inventory management, and artificial intelligence.

NAOUFEL CHEIKHROUHOU (Senior Member, IEEE) received the degree from the National School Engineers of Tunis, and the Ph.D. degree in industrial engineering from the Grenoble Institute of Technology, in 2001. He is currently a Professor of supply chain, logistics, and operations management with the Geneva School of Business Administration, University of Applied Sciences Western Switzerland. He was leading the Operations Management Research Group, Swiss Federal Institute of Technology at Lausanne (EPFL). He has published more than 100 papers in scientific journals and conferences and regularly acts as a keynote speaker in international academic and industrial conferences. Leading different projects with the collaboration of national and international industrial companies. His research interests include modeling, simulation and optimisation of enterprise networks, behavioral operations management, and demand planning and forecasting. He received the Burbidge Award, in 2003, and the BG Ingénieurs Conseils Award, in 2013, for his contribution to the research excellence in the fields.

NICOLAS ZUFFEREY received the B.Sc. and M.Sc. degrees in mathematics from the Swiss Federal Institute of Technology at Lausanne (EPFL), and the Ph.D. degree in operations research, in 2002. He was successively a Postdoctoral trainee with the University of Calgary, from 2003 to 2004, and an Assistant Professor with Université Laval, from 2004 to 2007. Since 2008, he has been with the University of Geneva, Switzerland, where he is currently a Full Professor of operations management. He is a member of the CIRRELT Transportation and Logistics Research Center and the GERAD Decision-Analysis Research Center. He is the coauthor of more than 140 publications (papers in academic journals, conference proceedings, and book chapters), and has reviewed articles for 54 international journals. With 76 coauthors, he has research activities with 32 Universities (mainly in Europe and America), and with 26 private companies. His research interests include designing solution methods for difficult and large optimization problems, with applications mainly in transportation, scheduling, production, inventory management, network design, supply chain management, and telecommunications.