Hybrid Algorithms for Designing Production Lines

Thesis submitted to
the Geneva School of Economics and Management,
University of Geneva, Switzerland

by

Khelil Kassoul

in fulfillment of the requirements for the degree of
PhD in Management

Members of the thesis committee:
Prof. Nicolas Zufferey, advisor, University of Geneva
Prof. Naoufel Cheikhrouhou, advisor, University of Applied Sciences Western Switzerland
Prof. Thomas Fischer, Chair, University of Geneva
Prof. Olivier Gallay, University of Lausanne

Thesis n° 112
October 2022

Hybrid algorithms for designing production lines

La Faculté d'économie et de management, sur préavis du jury, a autorisé l'impression de la présente thèse, sans entendre, par-là, émettre aucune opinion sur les propositions qui s'y trouvent énoncées et qui n'engagent que la responsabilité de leur auteur.

Genève, le 5 octobre 2022

Dean
Markus MENZ

Impression d'après le manuscrit de l'auteur

Hybrid algorithms for designing production lines

Acknowledgements

I would like to thank a number of people for their help and support during the production of this thesis.

First of all, I would like to express my sincere gratitude to my advisor Prof. Nicolas Zufferey, for giving me the opportunity of accomplishing this thesis, and for his guidance during all these years. I am especially grateful for his guidance, support with full encouragement and enthusiasm. I appreciated his assistance for writing, his great availability and his nice personality. I would like to express my deepest thanks to my co-advisor, Prof. Naoufel Cheikhrouhou, for his noble patience, guidance and full support. I have benefited greatly from his wealth of knowledge and meticulous editing. I am extremely grateful to him for his faith in me over these years and showing me the way to do my thesis. Thanks a lot, you both.

I am very grateful to my co-author Prof. Samir Brahim Belhaouari for his valuable advises, knowledge, and ideas.

I want to thank my thesis committee members: Prof. Olivier Gallay for his feedback (thanks for taking time to read my long thesis), and Prof. Thomas Fischer for agreeing to be chair of the jury.

Many thanks to my friend, Nabil Benothman, for his invaluable advices and assistance in programming codes. I gratefully recognize his help.

Finally, and most importantly, I am grateful for my family's unconditional, unequivocal, and loving support.

In memory of my parents: to my father who always encouraged me to do my thesis and to my mother who was always there for me from the first day of my birth until the last day of her life. For both of them, I love you and I miss you so much.

Abstract

This thesis proposes sophisticated simulation-based optimization approaches for the design of unreliable serial production lines. Two challenging problems are considered: the buffer allocation problem, and the joint buffer and service time allocation problem. Allocating optimal buffer capacities and the assignment of appropriate service time to each machine is fundamental to enhance the production rate and to reduce investment costs (either in the form of work-in-process inventory costs, floor space use or capital investment), and are then among the most essential insights needed for designing production lines. Therefore, the design of effective optimization techniques is essential for the competitiveness of enterprises.

The thesis is divided into four chapters. Chapter 1 and 2 aim to develop metaheuristics based on the particle swarm optimization. The algorithms proposed are developed to solve theoretical function, real optimization engineering, and the buffer allocation problem. Chapter 3 presents a robust hybrid technique, combining finite perturbation analysis with genetic algorithm, to solve the buffer allocation problem. The hybrid algorithm is extended in chapter 4 for allocating simultaneously service times and buffer capacities in production lines with unreliable machines.

Résumé

Cette thèse propose des approches d'optimisation sophistiquées basées sur la simulation pour la conception de lignes de production non fiables. Deux problèmes complexes sont considérés : le problème d'allocation des capacités de stockage tampon et le problème d'allocation des temps de service. Ces deux allocations sont fondamentales pour améliorer le taux de production et réduire les coûts d'investissement (soit sous la forme de coûts d'inventaire des travaux en cours, d'utilisation de l'espace au sol ou d'investissement en capital), et sont alors parmi les connaissances les plus essentielles nécessaires à la conception de lignes de production. Par conséquent, la conception de techniques d'optimisation efficaces est primordiale pour la compétitivité des entreprises.

La thèse est divisée en quatre chapitres. Les chapitres 1 et 2 visent à développer des métaheuristiques basées sur l'optimisation par essaim de particules. Les algorithmes proposés sont développés pour résoudre des fonctions théoriques, des problèmes réels d'ingénierie et le problème d'allocation des capacités de stockage. Le chapitre 3 présente une technique hybride couplant l'analyse de perturbations finies avec l'algorithme génétique pour résoudre le problème d'allocation des capacités de stockage. L'algorithme hybride est ensuite étendu, dans le chapitre 4, pour allouer simultanément les temps de service et les capacités de stockage dans les lignes de production avec des machines non fiables.

Contents

Abstract **6**
Résumé **7**
Contents **8**
Introduction **12**
1. Exponential Particle Swarm Optimization for global optimization **15**
1.1 Introduction 15
1.2 Related work 16
1.2.1 Introduction and control of parameters 17
1.2.1.1 The inertia weight 18
1.2.1.2 The constriction factor 18
1.2.1.3 Control of the parameters 19
1.2.2 The use of new strategies and neighborhood topologies 21
1.2.3 Hybrid strategies 25
1.3 Proposed exponential particle swarm optimization algorithm 26
1.3.1 The leaping strategy 28
1.3.2 The dynamic parameter and velocity controller 30
1.4 Experimental results 32
1.4.1 Experimental settings 33
1.4.2 Results on benchmark functions 35
1.4.2.1 Standard PSO with different parameters 35
1.4.2.2 ExPSO on benchmark problems 36
1.4.3 Convergence analysis 40
1.4.4 Algorithm Complexity 44
1.4.5 Statistical analysis 44
1.5 Results on engineering problems 45
1.5.1 Pressure Vessel Design 46
1.5.2 Compression spring design 47
1.5.3 Welded beam design 49
1.5.4 Speed reducer design 50
1.6 Conclusion 52
1.7 References 54

2. Adaptive Dynamic Jumping Particle Swarm Optimization for Buffer Allocation in Unreliable Production Lines 58
2.1 Introduction 58
2.2 Literature review 59
2.3 Problem formulation 62
2.4 Solution method 63
2.4.1 Particle Swarm Optimization (PSO) 63
2.4.2 Parameters and generation of the initial swarm of APSO 64
2.4.3 Proposed APSO 64
2.4.3.1 Logarithmic component: DCS-PSO 65
2.4.3.2 Exponential component: ExPSO 65
2.4.3.3 Expo-logarithm algorithm: APSO 68
2.5 Numerical Experiments 70
2.5.1 Results on a small instance with 3 machines 72
2.5.2 Results on a small instance with 5 machines 72
2.5.3 Results on a small instance with 10 machines 73
2.5.4 Results on medium instances with 20 machines 73
2.5.5 Results on large instances with 40 machines 74
2.5.6 Results on a large instance with 100 machines 75
2.5.7 Discussion and managerial insights 75
2.6 Conclusion 76
2.7 References 77
3. Buffer allocation design for unreliable production lines using Genetic Algorithm and Finite Perturbation Analysis 81
3.1 Introduction 81
3.2 Presentation of the problem and related work 82
3.2.1 Problem description and formulation 82
3.2.2 Literature review 83
3.3 Problem solving framework 86
3.4 Development of the optimization technique 88
3.4.1 Genetic algorithm (GA) 88
3.4.2 Finite perturbation analysis (FPA) 91
3.4.2.1 Perturbation generation rules in a unitary cell 93
3.4.2.2 Perturbation propagation rules in a unitary cell 94
3.4.3 Description of the FPA and SA algorithms 96

3.4.3.1 Notation 96
3.4.3.2 FPA algorithm 96
3.4.3.3 Stochastic Algorithm (SA) 97
3.5 Numerical Experiments 99
3.5.1 Determination of the parameters of the methods 99
3.5.2 Comparison of the proposed method with respect to the literature 101
3.5.1.1 Results for a 3-machine production line 102
3.5.1.2 Results for a 5-machine production line 103
3.5.1.3 Results for a 10-machine production line 103
3.5.1.4 Results for a 20-machine production line 104
3.5.3 Benefit of combining GA and FPA 105
3.5.4 Sensitivity analysis 106
3.6 Conclusion 108
3.7 References 109
4. Simultaneous allocation of buffer capacities and service times in unreliable production lines 113
4.1 Introduction 113
4.2 Literature review 114
4.3 Problem formulation and solution approach 117
4.3.1 Problem formulation 117
4.3.2 Solution approach 118
4.4 Development of the optimization technique 120
4.4.1 Genetic algorithm (GA) 120
4.4.1.1 Background 120
4.4.1.2 Selection operator 121
4.4.1.3 Crossover operator 121
4.4.1.4 Mutation operator 121
4.4.2 Finite perturbation analysis (FPA) 122
4.4.2.1 Case of starvation (NI) 124
4.4.2.2 Case of potential starvation (PNI) 124
4.4.2.3 Case of blocking (FO) 125
4.4.2.4 Case of potential blocking (PFO) 125
4.4.3 FPA algorithm 126
4.4.3.1 FPA algorithm for buffer capacities 126
4.4.3.2 FPA algorithm for service times 127
4.4.4 Stochastic Algorithm (SA) 128

4.5 Experiments 129
4.5.1 Comparison of GA, PA, and GA-FPA 130
4.5.2 Benefit of optimizing simultaneously buffers and service times 132
4.5.3 Solution patterns obtained by GA-FPA 132
4.5.4 Sensitivity analysis 137
4.5.4.1 Impact of having a machine with a different repair time 137
4.5.4.2 Impact of having a machine with a different failure time 139
4.6 Conclusion 140
4.7 References 141
Conclusion **144**

Introduction

In a world where demand for products is continually increasing, the design of serial production lines (which are widely employed in manufacturing systems) is a crucial problem in the field of production management and planning. Indeed, designing better production systems may result in higher output, efficiency, and quality standards. Even though the design and management of production systems have been studied in the literature for a while, they are still relevant and hot topics due to changing market trends, globalization, the constant desire to reduce production costs, and scientific and technological innovation. In such a context, this thesis has been carried out to develop efficient simulation-optimization tools and to propose relevant practical insights for designing serial production lines. The employed optimization techniques rely on *Particle Swarm Optimization* (PSO) and *Genetic Algorithms* (GA), which are well-known nature-inspired metaheuristics. Moreover, *Finite Perturbation Analysis* (FPA) is used as a local search mechanism in some of the proposed methods.

Serial production lines are often composed of several machines in which material (parts or items) flow from outside the system to the first machine M_1, then move to the first buffer b_1, then to machine M_2 and so on until they reach the last machine M_n and leave the production system. Figure 1 presents an example with n machines $(M_1, M_2, \ldots, M_n)$ connected between them and separated by $(n-1)$ buffers $(b_1, b_2, \ldots, b_{n-1})$, where b_i is the buffer capacity situated between two adjacent machines M_i and M_{i+1}. The sum of the employed buffer capacities is upper bounded.

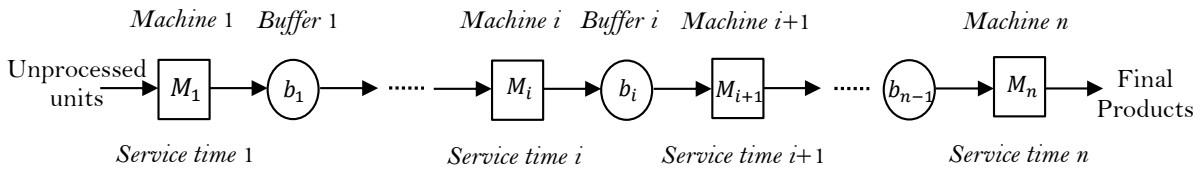


Figure 1. A serial production line

Two challenging manufacturing optimization problems are addressed in this thesis, with the aim of maximizing the production rate (i.e., the throughput).

- The *Buffer Allocation Problem* (BAP) consists of selecting the appropriate buffer allocation for the line. More precisely, the size b_i of each buffer i must be determined.
- The *Buffer and Service Time Allocation Problem* (BSTAP) consists of simultaneously determining the service time of the machines and the buffer capacities between them.

The two above problems are well-known NP-hard combinatorial-optimization problems. They can be solved for small instances (i.e., when the number n of machines is typically up to 10) with exact methods or commercial solvers. Such methods are unfortunately not appropriate for larger instances because of the increased computational complexity. For large instances (which usually characterize real-world applications), metaheuristics such as PSO and GA appear as efficient solution methods. Moreover, most of the literature on such problems concern reliable machines. In this thesis, only unreliable machines (e.g., breakdowns can occur) are considered, and a simulation tool has thus to be used. On the one hand, the obtained results will show the efficiency of the proposed methods, and on the other hand, they will provide insights for the production-line designers.

The thesis consists of four chapters. Each of them is an independent paper that has been published or submitted for publication in a refereed international journal. Each chapter presents the following features: scientific contributions, problem formulation, literature review, solution methods, computational experiences, conclusions, and possible future works.

- **Chapter 1** proposes an extension of the standard PSO algorithm for global optimization. Such a new method is called *Exponential Particle Swarm Optimization* (ExPSO), and it is characterized by a good balance between exploration (i.e., ability to visit various zones of the solution space) and exploitation (i.e., ability to focus and deeply investigate specific, promising regions of the solution space). ExPSO has been successfully tested over 29 benchmark problems, and 4 real optimization engineering problems. This chapter is published as a paper in *IEEE Access*.
- **Chapter 2** proposes an adaptive ExPSO method (denoted as APSO) to solve the BAP. Extensive numerical experiments for instances ranging from 3 to 100 machines are conducted. Comparisons are made with benchmark algorithms from the literature. This chapter is currently under evaluation in an international journal.
- **Chapter 3** combines GA and FPA to solve the BAP. Unlike the infinitesimal perturbation analysis, which deals with small (infinitesimal variation) perturbations for estimating gradients of the performance measure, FPA deals with larger (finite) or

more lasting perturbations. It is an extension specifically dedicated to discrete decision variables and applicable to most discrete-event dynamic systems. The role of GA is to diversify the search in the solution space (exploration), whereas the role of FPA is to intensify the search (exploitation). This chapter is published as a paper in *International Journal of Production Research.*

- **Chapter 4** extends GA-FPA to solve the BSTAP. The numerical experiments show that the obtained allocation patterns are in line with the literature for small instances, but important differences are highlighted for medium and large instances. Such observations provide new insights for decision makers for designing production lines. This chapter is accepted for publication in *International Journal of Production Research.*

Chapter 1

Exponential Particle Swarm Optimization for global optimization

1.1 Introduction

In the last years, the field of optimization has received significant attention as a promising tool for decision-making. In science, business, engineering, and other fields, there are many practical optimization problems. This chapter focuses on the single-objective optimization problem, and thus the global objective problem will be addressed as follows:

$$\begin{aligned} &\text{Minimize} && f(x) \\ &\text{Subject to} && l_i^b \le x_i \le u_i^b \,, i = 1, 2, \dots n \end{aligned} \tag{1.1}$$

where $x = (x_1, x_2, \dots, x_n)$ is a vector of n decision variables and defined by the bound constraints $l_i^b \le x_i \le u_i^b \,, i = 1, 2, \dots n$, and $f(x)$ is a continuous real objective function. The main objective is to converge towards a global optimum of the objective function *f*. Depending on the nature of the objective function and its properties, the complexity of the problem can vary enormously.

For solving such problems, researchers have presented many algorithms in past years. Some methods are inspired by the social and biological behavior of natural colonies, like bird flocks or fish schools. This group names the swarm intelligence (SI) techniques such as Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995), Harris Hawks Optimization (HHO) (Heidari et al., 2019), Whale Optimization Algorithm (WOA) (Mirjalili and Lewis, 2016), Ant Colony Optimization (ACO) (Mahi et al., 2015), Artificial Bee Colony (ABC) (Z. Li et al., 2015), and Grey Wolf Optimizer (GWO) (Mirjalili et al., 2014).

PSO is one of the most robust and effective SI algorithms and is widely used due to its simplicity and adaptability to a various application. It achieves a large performance satisfaction in solving many kinds of problems in global optimization (Rauf et al., 2020),

constraints optimization (J. Yang et al., 2019), and multi-objective optimization (Raza et al., 2018), but convergence is not always guaranteed (premature convergence).

This chapter investigates the performance of the PSO and proposes a novel variant of PSO, named Exponential Particle Swarm Optimization (ExPSO) for global optimization problems by introducing a leaping strategy based on exponential function and dynamic parameters.

The remainder of the chapter is organized as follows. Section 1.2 introduces related works. The proposed PSO approach (ExPSO) is presented in Section 1.3. Its application on 29 classical benchmark functions is discussed in Section 1.4, and its application on constrained real-world optimization problems is presented in Section 1.5. Finally, a conclusion and future research directions are proposed in Section 1.6

1.2 Related work

Kennedy and Eberhart (1995) introduce an algorithm called Particle Swarm Optimization (PSO) which is inspired by the social collective behavior of bird flocks or fish schooling. PSO is a global optimization heuristic, where each bird (called particle) forms a potential solution in the search space. During the repeated flights, each particle learns from the other particles and memorizes the global best position (g_{best}), i.e., the best solution visited by any particle of the swarm. The particle learns also from its own experience and memorizes its personal best solution visited so far (p_{best}). PSO is an iterative method that starts with a set of random solutions and then it searches for optima by updating the iterations. Figure 1.1 shows an example where each particle considers the previous velocity, the global best position (social part), and the personal best position (cognitive part), to determine the new velocity and update its position.

PSO can be formally described as follows. Assume that the search space is d–dimensional, where each particle (a potential solution) has several parameters such as the current position x, the velocity v, the personal best position p_{best} visited by the particle so far, and the global best position g_{best} that any particle in the swarm has experienced. The position and the velocity of the i^{th} particle at iteration t are represented by $x_i^t = (x_{i1}, x_{i2}, \dots, x_{id})$ and $v_i^t = (v_{i1}, v_{i2}, \dots, v_{id})$, respectively. At every iteration t, each particle is updated by p_{best} and g_{best}. The velocity and position, in the original version of PSO (Kennedy and Eberhart, 1995), are updated as follows:

$$v_i^{t+1} = v_i^t + c_1 r_1 \left({p_{best}}_i^t - x_i^t \right) + c_2 r_2 (g_{best}^t - x_i^t) \tag{1.2}$$

$$x_i^{t+1} = v_i^{t+1} + x_i^t \tag{1.3}$$

where ${p_{best}}_i^t$ denotes the *personal best* position of the i^{th} particle, g_{best}^t denotes the *global best* position of the whole swarm at the iteration *t*, the two acceleration constants c_1 *and* c_2 are known respectively as *cognitive* and *social* scaling parameters, and r_1 and r_2 are two random values in the interval $[0,1]$.

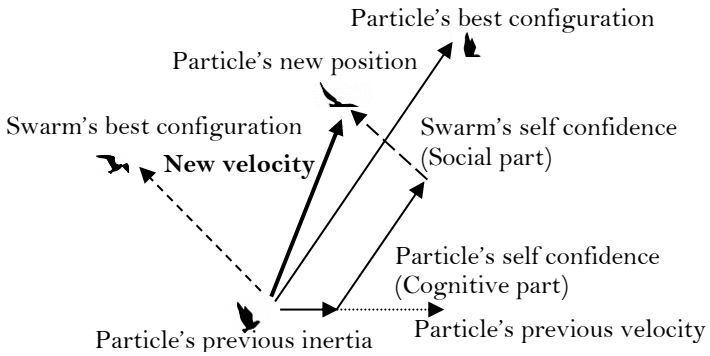


Figure 1.1. New positions process

Since its introduction, PSO has undergone several modifications such as the introduction of a factor called *inertia weight* (Shi and Eberhart, 1998), and other versions have been proposed such as the proposition of new strategies (Rehman et al., 2020; Zhang et al., 2020a). This is because the basic PSO has its limitations, including premature convergence, and it is designed just for real optimization. Although other parameters have been introduced or modified, and different aspects of the original PSO have been changed, the addition of other optimization algorithms has been proposed, and thus many variants of PSO have been developed to solve different types of problems. Consequently, the algorithm can be applied to several fields.

The following subsections present some instances where the basic PSO algorithm has been modified. First, Subsection 1.2.1 introduces new parameters integrated into PSO. Next, the use of new strategies and neighborhood topologies is discussed in Subsection 1.2.2. Finally, Subsection 1.2.3 presents hybrid strategies of PSO with other commonly used metaheuristics.

1.2.1 Introduction and control of parameters

To obtain better convergence, modifications of the standard PSO have been proposed. A critical factor in the success of PSO is the balance between local and global search. In this context, different works have been conducted to control the velocity and tune the parameters of the PSO.

1.2.1.1 The inertia weight

Shi and Eberhart (1998) improve the method by introducing the inertia weight "w". This factor has the role of ensuring, throughout the entire search space, a compromise between local and global search. It is shown that particles tend to search for global solutions if the inertia weight takes a value from the interval [0.8,1.2], and if the value of this parameter is small, often particles converge prematurely towards local optima. The modified PSO with the inertia weight is commonly referred to as the Standard PSO. Thus, the velocity is updated as follows:

$$v_i^{t+1} = wv_i^t + c_1 r_1 \left(p_{best_i}{}^t - x_i^t\right) + c_2 r_2 (g_{best}^t - x_i^t) \tag{1.4}$$

Since the introduction of the inertia weight, different studies propose to improve this parameter. Nickabadi et al. (2011) present different inertia weight models including constant, adaptive inertia weight, and time-varying strategies based on the success rate of the particles. Recently, Taherkhani and Safabakhsh (2016) propose an adaptive inertia-weight strategy based on the performance particles and the distance from their best position. Ardizzon et al. (2015) dynamically adjust the inertia weight at each iteration related to the distance of each particle from the global best position and also test the influence of the acceleration coefficients on the convergence of the method.

1.2.1.2 The constriction factor

To ensure the convergence of the PSO, Clerc (1999) introduces a new parameter called the constriction factor "χ", where, as the search progresses, it reduces the effect of the particle's velocity. The suggestion of using the constriction factor is to ensure a trade-off between exploitation (i.e., the ability to intensify the search in solution-space regions currently investigated) and exploration (i.e., the ability to explore diversified solution-space regions). This factor is given by:

$$\chi = \frac{2}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|} \quad , \quad \varphi = \varphi_1 + \varphi_2 \tag{1.5}$$

Under the constraint that $\varphi > 4$, φ_1 and φ_2 can be split into products of random values r_1 and r_2 times the social and cognitive acceleration coefficients c_1 and c_2. Generally, the value of φ is 4.1 for $\varphi_1 = \varphi_2 = 2.05$, which gives a value of $\chi \approx 0.7298$. The velocity is therefore updated as follows:

$$v_i^{t+1} = \chi[v_i^t + \varphi_1 r_1 \left(p_{best_i}{}^t - x_i^t\right) + \varphi_2 r_2 (g_{best}^t - x_i^t)] \tag{1.6}$$

In equation 1.7, Clerc and Kennedy (2002) introduce an additional parameter $m \in [0,1]$ in the original formula of the constriction factor to control the convergence speed of particles, i.e., finding a balance between the global and local search of the swarm. If *m* is set close to 0, a fast convergence is obtained but results in a low degree of accuracy in the solution quality. In contrast, when *m* is close to 1, slow convergence and high diversification degree in finding the best solution ensues.

$$\chi = \frac{2m}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|} \quad , m \in [0,1] \tag{1.7}$$

The use of the constriction coefficient can be viewed as a recommendation to control the dynamic characteristics of the particle swarm, and it thus gives better quality solutions.

1.2.1.3 Control of the parameters

Molaei et al. (2021) develop a new PSO variant using a crossover operator for solving mathematically generated landscapes. The proposed method includes a parameter updating scheme, an enhanced learning strategy, and a crossover operator for diversifying the swarm. In their study, the authors define a separate self-cognition coefficient per particle, and give a self-cognition vector $C_1^t = [c_{11}^t, c_{12}^t, \dots, c_{1N}^t]$ as:

$$C_1^t = F^t \cdot R^t \tag{1.8}$$

where $F^t = [f_1^t, f_2^t, \dots, f_N^t]$ is the normalized fitness values of the particles, *N* is the number of particles and R is a nonnegative random matrix with elements $\in [0,1]$. In each iteration, the social acceleration parameter takes the value of the largest value of the cognitive parameter among all the particles. Thus, the *social* parameter c_2 is defined as:

$$c_2^t = \max_i(c_{i1}^t) \ , 1 \le i \le N \tag{1.9}$$

The velocity in the proposed algorithm is updated as follows:

$$v_i^{t+1} = wv_i^t + \sum_{j=1}^{n} c_{1\mathrm{j}}^t \left(p_{best}{}_j^t - x_i^t\right) + c_2^t r_2 (g_{best}^t - x_i^t) \tag{1.10}$$

Wang et al. (2019) propose another PSO variant to optimize the hyper-parameter configuration of Convolutional Neural Network (CNN) using compound normal distribution to improve the exploration ability of the standard PSO. They vectorize the *cognitive* and *social* scaling parameters c_1 *and* c_2. Thus, the velocity is updated as follows:

$$v_i^{t+1} = wv_i^t + c_1^d r_1 \left(p_{best}{}_i^t - x_i^t\right) + c_2^d r_2 (g_{best}^t - x_i^t) \tag{1.11}$$

where, c_1^d and c_2^d are two d-dimensional vectors.

Ghasemi et al. (2019) propose a primary variant of PSO called phasor particle swarm optimization (PPSO) which replaces the control parameters of the algorithm ($c_1 r_1$) and ($c_2 r_2$) in the original PSO with a scalar phasor angle (θ) based on trigonometric functions. The velocity of each particle is given by:

$$v_i^t = p(\theta_i^t)\left(p_{best_i}{}^t - x_i^t\right) + g(\theta_i^t)(g_{best}^t - x_i^t) \tag{1.12}$$

where, the inertia factor w is set to zero and where $p(\theta_i^t)$ and $g(\theta_i^t)$ are determined as:

$$p(\theta_i^t) = |cos\theta_i^t|^{2sin\theta_i^t} \tag{1.13}$$

$$g(\theta_i^t) = |sin\theta_i^t|^{2cos\theta_i^t} \tag{1.14}$$

Fielding and Zhang (2020) develop a PSO-based algorithm for image classification to optimize DenseBlock CNN architecture. The proposed method employs adaptive acceleration coefficients generated using cosine annealing learning rate schedule to overcome local optima traps. Therefore, the acceleration coefficients are defined as:

$$c_1^t = b + \frac{a-b}{2}\cos\left(\pi\left(1 - \frac{t}{T}\right)\right) + 1 \tag{1.15}$$

$$c_2^t = b + \frac{a-b}{2}\cos\left(\pi\frac{t}{T}\right) + 1 \tag{1.16}$$

where, a and b denote the initial and final learning rate values, respectively, and T represents the maximum numbers of iterations.

However, in the study of Yalcinoz and Rudion (2019), the cognitive and social scaling parameters c_1 and c_2 are taken as function and recalculated at each iteration t as follows:

$$c_1^t = c_2^t = 2 + \frac{t_{max} - t}{5t_{max}} \tag{1.17}$$

where *tmax* denotes the maximum allowed number of iterations.

Kassoul et al. (2021) introduce new cognitive and social functions $c_1(t)$ and $c_2(t)$. They integrate a logarithm function into the function $c_1(t)$ as follows:

$$c_1^t = (a^t)^\alpha \cdot log\left(\left\|p_{best_i}{}^t - x_i^t\right\| + \epsilon\right) \tag{1.18}$$

where, $a(t)$ is a dynamic cognitive parameter, x_{max} and x_{min} are the upper and lower bounds of variables, respectively, ϵ is a very small positive value, and α is a dynamic regulator exponent defined as follows:

$$\alpha = \begin{cases} 5\,, & |x_{max} - x_{min}| \leq 10 \\ 1\,, & |x_{max} - x_{min}| > 10 \end{cases} \tag{1.19}$$

They test four initial values for the dynamic cognitive parameter (i.e., e, $\sin(\sqrt{\chi\pi})$, $\cos(\sqrt{\chi\pi})$ and $(\chi\pi)^{\frac{1}{4}}$), whereas the social function is taken as $c_2^t = (\chi\pi)^{\alpha}$.

1.2.2 The use of new strategies and neighborhood topologies

In the last decades, PSO has been widely studied with promising performance for different optimization problems. To enhance the performance of PSO, the following features have been often investigated: escape from local optima, avoid premature convergence, and improve exploration and exploitation capabilities.

Some PSO variants introduce different adaptive and learning strategies. Zhang et al. (2020b) propose a PSO algorithm with an adaptive learning strategy to strengthen the global search ability and increase the population diversity, thus helping the algorithm to escape from local optima. To increase the accuracy of the implementation of the learning strategy, the proposed method is based on an adaptive subswarm, where the whole swarm is adaptively clustered into several subswarms. The learning strategy for locally best particles is calculated by the equation (1.20).

$$x_i^d = x_i^d + c_1 r_1^d \left(p_{best_i}{}^d - x_i^d\right) + \; c_2 r_2^d \left(\frac{1}{C}\left(L_{best_c}{}^d - x_i^d\right)\right) \tag{1.20}$$

where x_i^d represents the vector position ($i = 1,2,\dots,d$), d is the dimension of the problem, and L_{best} is the locally best particle in subswarm c.

Xu et al. (2019) propose a dimensional learning strategy (DLS) that discovers and protects the information of the best solution of the population and develop a Two-Swarm Learning PSO algorithm (TSLPSO) based on different learning approaches to construct a learning pattern for each particle. They validate the performance of the algorithm on benchmark functions and a real-world engineering problem. Earlier, Ho et al. (2008) present the orthogonal particle swarm optimization applying an intelligent move strategy to adapt the particles' velocities. The mechanism aims to generate, for each particle, two temporary positions: $H = (h_1, h_2, \dots, h_d)$ corresponding to the cognitive part, and $T = (t_1, t_2, \dots, t_d)$ corresponding to the social part:

$$h_d = x_i^t{}_d + w v_i^t{}_d + c_1 r_1 \left(p_{best}{}_i^t{}_d - x_i^t{}_d\right) \tag{1.21}$$

$$t_d = x_i^t{}_d + w v_i^t{}_d + c_2 r_2 \left(g_{best}{}^t{}_d - x_i^t{}_d\right) \tag{1.22}$$

The goal is to combine the good partial vectors of H and T to obtain the next position x_i^{t+1}, and then the particle velocity is calculated as follows:

$$v_i^{t+1} = x_i^{t+1} - x_i^t \tag{1.23}$$

where x_i^{t+1} and v_i^{t+1} are the position and the velocity of the i^{th} particle at iteration t+1, respectively. The authors test the performance of their approach in solving large discrete task assignment problems and real-valued parametric functions.

Xia, Xing, et al. (2019) propose a Fitness-based Multi-role PSO (FMPSO) by integrating a new component (sub-social learning part) into the standard PSO to help the swarm to execute various search mechanisms and to take full advantages of local and global topologies. In the proposed algorithm, the velocity is updated using four parts instead of the three parts used in the standard PSO (i.e., the inertia weight, the cognitive and social scaling parameters). Accordingly, the equation of the velocity is described as:

$$v_i^{t+1} = wv_i^t + c_1 r_1 s_{i_{d1}}\left(p_{best_i}{}^t - x_i^t\right) + c_2 r_2 s_{i_{d2}}\left(g_{best}{}^t - x_i^t\right) + c_3 r_3 s_{i_{d3}}\left(l_{best_i}{}^t - x_i^t\right) \tag{1.24}$$

where l_{best} is the local (neighbors') best position, and $s_{i_{d1}}$, $s_{i_{d1}}$ and $s_{i_{d1}}$ are three objective dimension vectors of the i^{th} particle. In addition, individuals are classified based on their fitness in three different categories (leader, rambler and follower).

Recently, Xia et al. (2020) propose an expanded PSO (XPSO). Inspiring by the extraction of knowledge from the multi-exemplar and forgetting (or removing) unused memories in biological systems and human society, the authors integrate the forgetting ability and multi-exemplar to the standard PSO. They choose the global $g_{best}$and local best g_{best} positions as exemplars and assign for each particle a specific forgetting ability $F = \left(f_i^1, f_i^2, \ldots, f_i^d\right)$, where f_i^j represents the forgetting ability of how much information is forgotten by the j^{th} dimension for the i^{th} particle when learning from the exemplars. Consequently, the update of the i^{th} particle is given by:

$$v_i^{t+1} = wv_i^t + c_1 r_1\left(p_{best_i}{}^t - x_i^t\right) + c_2 r_2\left(\left(1 - f_i^j\right) g_{best}{}^t - x_i^t\right) + c_3 r_3\left(\left(1 - f_i^j\right) l_{best_i}{}^t - x_i^t\right) \tag{1.25}$$

Rehman et al. (2020) introduce the Multi-Cluster PSO for large-scale optimization, where the population of particles is divided into diverse clusters. They add two components to the standard PSO to deal with its limitations and use random and semi-random initialization of particles to escape from local optima by dividing the entire search space into equal intervals.

The length of each interval is given by:

$$L_{int}^{d} = \frac{(u_b - l_b)_d}{M} \tag{1.26}$$

where u_b and l_b are the lower and upper bounds of the search space, respectively, M is the swarm size, and d is the problem dimension. The velocity equation is updated as:

$$v_i(t+1) = \omega v_i(t) + c_1 r_1 \left(p_{best_i}{}^t - x_i^t\right) + c_2 r_2 \left(g_{best}{}^t - x_i^t\right) + c_3 r_3 \left(C_{best_i}{}^t - x_i(t)\right) + J_i(C\mathcal{J}) \tag{1.27}$$

where c_3 is a scaling factor and $C_{best_i}{}^t$ is the cluster best position for the i^{th} particle at the iteration t. The term $J_i(C\mathcal{J})$ is calculated as:

$$J_i(C\mathcal{J}) = \left(\log\left(\frac{\left(\frac{C\mathcal{J}_i}{0.1 * acceptance}\right)^{\alpha}}{\left(\frac{C\mathcal{J}_i}{0.1 * acceptance}\right)^{\alpha} + 1} + \varepsilon \right) \right) rand^d \tag{1.28}$$

where,

$$C\mathcal{J}_i = \|v_i(t)\| + \left\|p_{best_i}{}^t - x_i^t\right\| + \left\|g_{best}{}^t - x_i^t\right\| + \left\|C_{best_i}{}^t - x_i(t)\right\| \tag{1.29}$$

where α and ε are initial parameters. To test the performance of their approach, an extensive experimentation is carried out to optimize twelve high dimensional dynamic benchmark functions.

Gandomi et al. (2013) introduce chaos into the Accelerated PSO (APSO), developed by Yang (2010), to reveal whether chaotic maps are essential to enhance the global search ability of the PSO. APSO uses only the global best to accelerate the convergence of the algorithm. At each iteration, the velocity and the position are updated as follows:

$$v_i^{t+1} = v_i^t + \alpha r + \beta \left(g_{best}{}^t - x_i^t\right) \tag{1.30}$$

$$x_i^{t+1} = (1-\beta) x_i^t + \beta g_{best}{}^t + \alpha r \tag{1.31}$$

where r is generated randomly from a Gaussian distribution, $\alpha = 0.1L$ to $0.5L$ with L being the scale of each variable, and $\beta = 0.2$ to 0.7. The authors investigate twelve different chaotic maps and use three engineering problems to validate the performance of the method developed.

Xia, Gui, and Zhan (2018) develop a Multi-Swarm PSO (MSPSO) algorithm based on purposeful detecting strategy and dynamical topology. The authors propose three different strategies. First, the entire swarm is divided into many sub-swarms using a Dynamic Number

Strategy (DNS) which increases periodically the size of sub-swarm. Next, when the global best position stagnates, the Sub-swarm Regrouping Strategy regroup through each DNS period the entire population into new sub-swarms. Finally, a Purposeful Detecting Strategy is executed to avoid local minima using some historical information of the search process. The performance of the approach is tested on several commonly benchmark functions and four real applications.

In addition to the implementation of different learning strategies, research on neighborhood topologies to control exploration and exploitation – and reduce the loss of diversity – has also attracted considerable attention. The neighborhood or population topology plays an important role to enhance the performance of the PSO. The neighbors of each particle are determined by the topology to collect the information that each particle shares with its neighbors and learns from them. Two topologies are presented in the basic PSO: the local best topology (*lbest*), where a particle is connected only with its nearest neighbors; the global best topology (*gbest*), where each particle is a neighbor for all other particles. Kennedy and Mendes (Kennedy and Mendes, 2002) introduce six population architectures (wheel, star, von Neumann, all, ring, and random) presented in Figure 1.2. The authors confirm that the topology of PSO significantly affects the performance of the algorithm and recommend the use of the von Neumann topology. Mendes et al. (2004) use the information of all neighbors to control a particle in the swarm and test the performance of the fully informed particle swarm using different topologies.

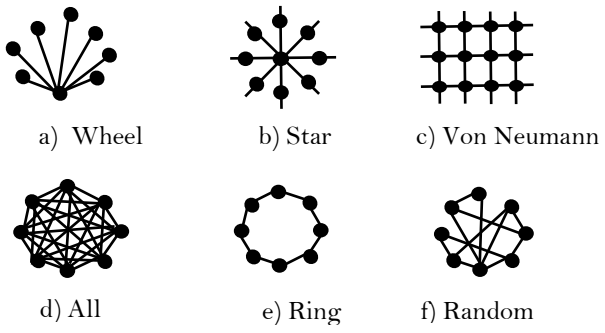


Figure 1.2. Topologies used in PSO

The reader can refer to Lynn et al. (2018) for a comprehensive review of different topologies developed for particle swarm optimization and differential evolution.

1.2.3 Hybrid strategies

To overcome the limitations of PSO (e.g., being trapped in local optima) and combine its advantages with the advantages of other algorithms, some researchers develop hybridized PSO and propose different variants of hybridization approaches. Table 1.1 presents a summary of some recent research on hybrid strategies

Table 1.1. Hybrid strategies

Author/s:	**year**	**Method**	**Hybridization with**	**Area of application**	**Some details**
Issa et al. (2022)	2022	BPINF	BA (Butterfly Algorithm)	Computational Biology (Case study: COVID-19)	Novel infection propagation mechanism. Fragmented local aligner technique.
El-Shafiey et al. (2022)	2022	GAPSO-RF	GA (Genetic Algorithm)	Feature selection and heart-disease prediction	Multivariate statistical analysis for the initial population. Discriminate mutation strategy.
Li et al. (2021)	2021	SA-PSO	SA (Simulated Annealing)	Design of power amplifiers	Elastic collision model to improve the searchability. A stepped fitness function is used to solve the performance pit in design.
Ghorpade et al. (2021)	2021	EDCQPSO	DE (Differential Evolution)	Nonlinear numerical problems and Internet of Things (IoT) applications	A quantum PSO, differential evolution operator, and crossover operator.
Rauf et al. (2021)	2021	CMHDE-PSO	DE (Differential Evolution)	Continuous optimization and classification problems	Novel distribution named Torus for the choice of initial population. New parameter adoption technique based on chaos theory.
Alam and Arefifar (2020)	2020	PSO-TS	TS (Tabu Search)	Power distribution system	Reciprocating strategy PSO-TS based on neighbor research.
Dabhi and Pandya (2020)	2020	EVDEPSO	DE (Differential Evolution)	Power systems in microgrid environment	Mutated inertia weight and new strategic parameters.
Pitchaimanickam and Murugaboopathi (2020)	2020	HFAPSO	FA (Firefly Algorithm)	Wireless Sensor Networks	Finding the energy optimal cluster selection in wireless sensor networks Reducing the energy consumption and prolong the network lifetime.
Mashwani et al. (2020)	2020	ESEA	DE, BA, TLBO (Teaching Learning-Based Optimization)	Global Optimization	Employ of a self-adaptive process to progress randomly generated set of solutions.
Tang et al. (2019)	2019	HDPSO-SA	SA (Simulated Annealing)	Engineering design optimization	Design of 4 neighborhood structures. Novel global best selection method.
Ullah et al. (2019)	2019	PPSOGSA	GSA (Gravitational Search Algorithm)	Solar photovoltaic generators and power systems	A phase angle (θ) is used to model the particle control parameters.
Chi et al. (2019)	2019	CSPSO	CS (Cuckoo search)	Engineering design and continuous functions problems	Use orthogonal Lation squares to construct the initial positions of the CS. Dynamic adjustment for the step size of the CS.
Nagra et al. (2019)	2019	GSADMSPSO	GSA (Gravitational Search Algorithm)	Complex optimization problems	New neighborhood strategy and dynamic multi swarm configurations.
Xia et al. (2018)	2018	FAPSO	FA (Firefly Algorithm)	Global Optimization	Selection of two equal sub-populations for PSO and FA. Integration of a detecting operator to exploit promising subregions. Introduction of BFGS Quasi-Newton method.

Sedighizadeh and Mazaheripour (2018)	2018	PSOABCC	ABC (Artificial Bee Colony)	Vehicle routing and traveling salesman problems	PSO phase and ABC phase methods are taken in parallel to improve the personal bests.
Benvidi et al. (2017)	2017	ANN-PSO-GA	GA (Genetic Algorithm)	Spectrophotometric of food colorants	Use of a sensitive and selective procedure for the correct prediction of concentrations.
Javidrad and Nazari (2017)	2017	PSO-SA	SA (Simulated Annealing)	Global optimization and engineering design	A reciprocating mechanism is used between SA and PSO. Three criteria (method) for comparative work.
Garg (2016)	2016	PSO-GA	GA (Genetic Algorithm)	Constrained optimization problems and engineering design	GA used to modify decision vectors, whereas PSO is used to improve them.

Concerning the existing literature, the main contributions of this chapter are the following.

- A new search strategy using an exponential function is proposed, permitting particles to make leaps in the search space. The swarm population is divided into subpopulations to efficiently explore the whole search space. This leap strategy is integrated into the equation of the velocity updating, allowing thus to escape from local optima.
- Personal and global worst positions are incorporated in the search process to maintain population diversity by moving against their positions to escape from eventual local minima, allowing thus the exploration of new regions of the search space.
- A concept of iteration division is employed to balance the diversity of the population and the intensification process.
- A new linear decreasing cognitive parameter with a dynamic inertia weight strategy is integrated into the proposed method. The intention is to explore the entire search space by permitting larger leaps at the beginning of the search, and then smaller jumps for improvement in particular space regions.
- New values of cognitive and social scaling parameters of the standard PSO are investigated.

The following section explains the details of the proposed algorithm.

1.3 Proposed exponential particle swarm optimization algorithm

This section presents in detail the proposed Exponential Particle Swarm Optimization Algorithm (ExPSO) which includes two primary purposes. First, to avoid falling into premature convergence, a leaping strategy is developed. The second objective is to introduce a dynamic parameter along with controlling the velocity to converge quickly and reliably towards the best solution. A detailed flowchart for ExPSO is presented in Figure 1.3.

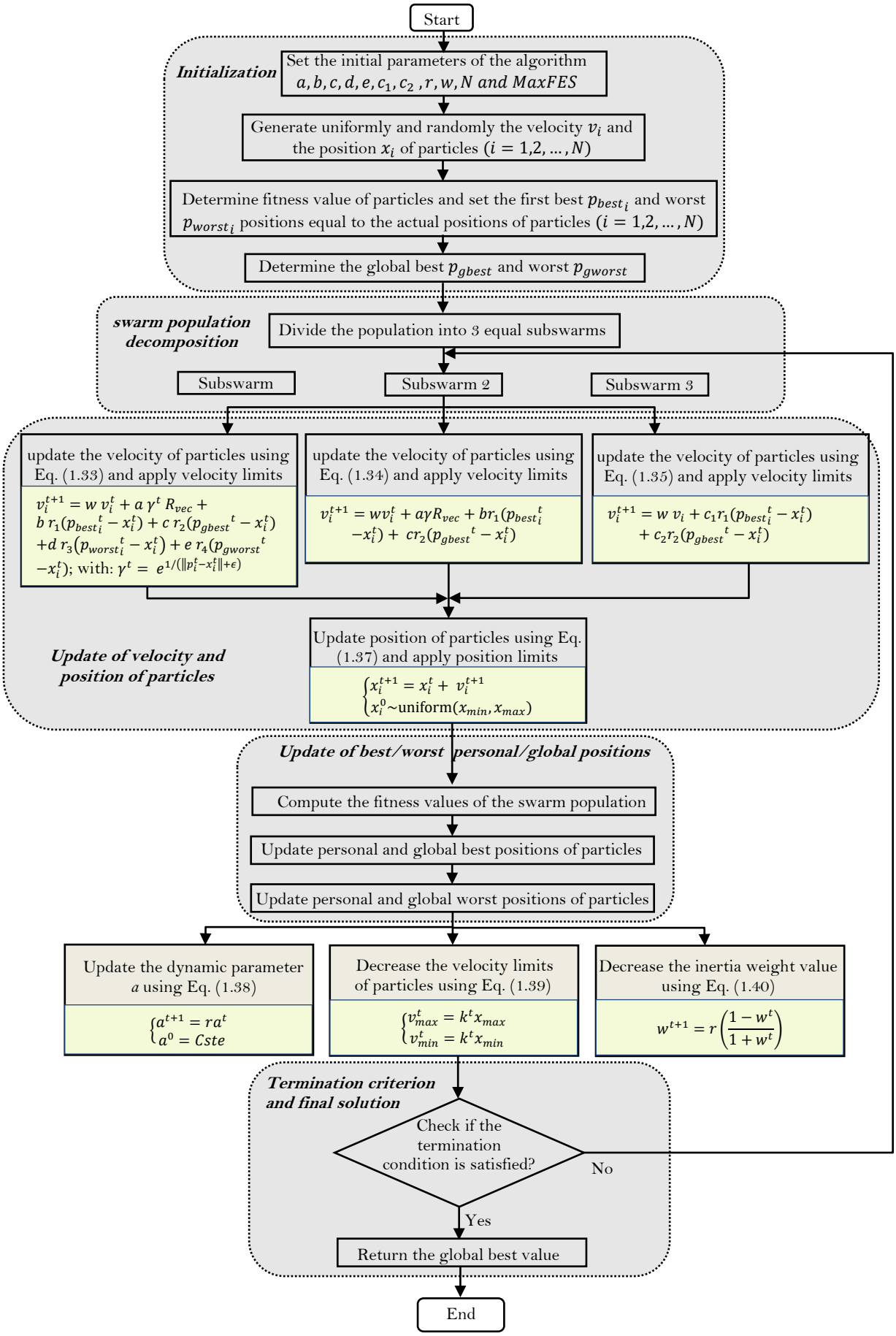


Figure 1.3. Flowchart of the proposed ExPSO

1.3.1 The leaping strategy

The proposed method is characterized by dividing the swarm into subpopulations to explore better the whole search space. First, the algorithm begins by initializing a population of N particles, concerning the search range of the problem, uniformly and randomly. Next, the population of particles is divided into 3 equal subpopulations N_1, N_2 *and* N_3. The velocity of each particle in the first subpopulation N_1 is ruled by the previous velocity, the personal best position p_{best}, the global best position g_{best}, the personal worst position p_{worst}, and the global worst position g_{worst}. The swarm's particles should follow the ones that have gained an enhancement in their respective positions, whilst the integration of the personal and global worst positions maintains a diversity of the population and explore new regions that could help overcome the issue of premature convergence. In addition, a leaping strategy is also integrated into the velocity equation to enable the particles to avoid local minima traps, using an exponential function γ^t defined at every iteration t as in equation (1.32), where ϵ is a very small positive value to avoid the division by zero.

$$\gamma^t = e^{1/(\|p_i^t - x_i^t\| + \epsilon)} \tag{1.32}$$

Using the exponential function is important when the velocity of a particle is near zero (i.e., the particle is trapped in an eventual local optimum). In this case, the particle could leap using the exponential function to avoid a possible local optimum. The velocity of the first subpopulation N_1 is thus updated as in equation (1.33), where w is the inertia weight, R_{vec} is a random vector, and $a, b, c, d,$ and e are parameters defined at the beginning of the run, respectively called in this chapter, the exponential weight parameter, cognitive acceleration best-coefficient, social acceleration best-coefficient, cognitive acceleration worst-coefficient, and social acceleration worst-coefficient.

$$\begin{aligned} v_i^{t+1} = w\, v_i^t + a\, \gamma\, R_{vec} + b\, r_1\left(p_{best_i}{}^t - x_i^t\right) + c\, r_2\left(p_{gbest}{}^t - x_i^t\right) \\ + d\, r_3\left(p_{worst_i}^t - x_i^t\right) + e\, r_4\left(p_{gworst}{}^t - x_i^t\right) \end{aligned} \tag{1.33}$$

To investigate new solution-space regions that might contain the global optimum, the cognitive acceleration worst-coefficient for the personal worst position and the social acceleration worst-coefficient for the global worst position have negative values (i.e., we investigate opposite regions to those followed by the personal and global worst positions). Figure 1.4 shows the jump made by the particle when it is blocked in an eventual local minimum (its velocity is near

to zero). This could help the particle to overcome premature convergence by exploiting other regions of the search space.

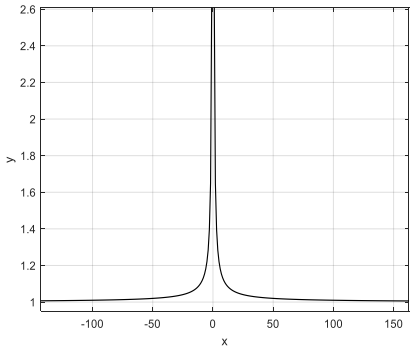


Figure 1.4. The leaping strategy ($f(x) = e^{\frac{1}{|x|}}$)

For the second subpopulation N_2, the velocity is influenced (in addition to the exponential function) by the previous velocity and the personal and global best positions. The velocity is updated here as:

$$v_i^{t+1} = wv_i^t + a\gamma R_{vec} + br_1\left({p_{best}}_i^t - x_i^t\right) + cr_2\left({p_{gbest}}^t - x_i^t\right) \tag{1.34}$$

Finally, for the rest of the particles of the swarm, the update of the velocity follows the procedure of C_NPSO (described in Subsection 1.4.2.1). The equation of the velocity, in this case, is given by:

$$v_i^{t+1} = w\, v_i + c_1 r_1\left({p_{best}}_i^t - x_i^t\right) + c_2 r_2\left({p_{gbest}}^t - x_i^t\right) \tag{1.35}$$

Algorithm 1.1 updates the procedure of the three subpopulations of the swarm.

Algorithm 1.1. Subpopulations

Set the value of N (*N is the population size*)

- $N_1 = N_2 = N_3 = N/3$ (*N_1, N_2 and N_3 are the number of particles of subpopulations 1,2 and 3, respectively*)

While (the maximum number of iterations is not reached), **do**

If $i \leq N_1$

- $\gamma = e^{1/(\|{p_{best}}_i - x_i\| + \epsilon)}$
- $v_i = wv_i + a\gamma R_{vec} + br_1\left({p_{best}}_i - x_i\right) + cr_2\left(p_{gbest} - x_i\right) + dr_3\left({p_{worst}}_i - x_i\right) + er_4\left(p_{gworst} - x_i\right)$

Else if $N_1 < i \leq N_1 + N_2$

- $v_i = wv_i + a\gamma R_{vec} + br_1\left({p_{best}}_i - x_i\right) + cr_2\left(p_{gbest} - x_i\right)$

else

- $v_i = wv_i + c_1 r_1\left({p_{best}}_i - x_i\right) + c_2 r_2\left(p_{gbest} - x_i\right)$

end while

The velocity equation of each particle in the proposed ExPSO is updated at every iteration t as:

$$v_i^{t+1} = \begin{cases} v_{max} & if \quad wv_i + c_1 r_1 \left(p_{best_i}{}^t - x_i^t\right) + c_2 r_2 \left(p_{gbest}{}^t - x_i^t\right) > v_{max} \\ v_{min} & if \quad wv_i + c_1 r_1 \left(p_{best_i}{}^t - x_i^t\right) + c_2 r_2 \left(p_{gbest}{}^t - x_i^t\right) < v_{min} \\ wv_i + c_1 r_1 \left(p_{best_i}{}^t - x_i^t\right) + c_2 r_2 \left(p_{gbest}{}^t - x_i^t\right) & otherwise \end{cases} \quad (1.36)$$

$$v_i^0 \sim \text{uniform}(v_{min}, v_{max})$$

The updated position follows the following sequence

$$\begin{cases} x_i^{t+1} = x_i^t + v_i^{t+1} \\ x_i^0 \sim \text{uniform}(x_{min}, x_{max}) \end{cases} \quad (1.37)$$

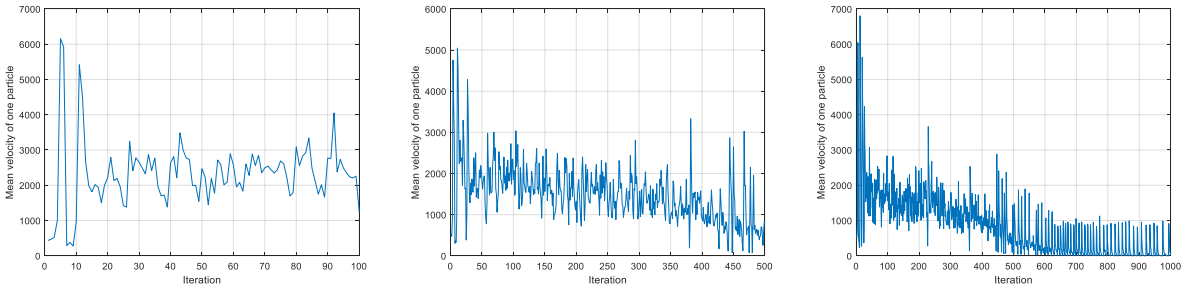
1.3.2 The dynamic parameter and velocity controller

To perform local and global search efficiently, exploitation and exploration phases must be well balanced. For this purpose, the proposed ExPSO uses a linear decreasing exponential weight parameter a to identify new regions within the search space. A damping coefficient $r \in [0.5,2]$ is used, allowing larger jumps to explore the entire search space at the start of the search, and smaller jumps for improvement in particular space regions. The equation of a is given by:

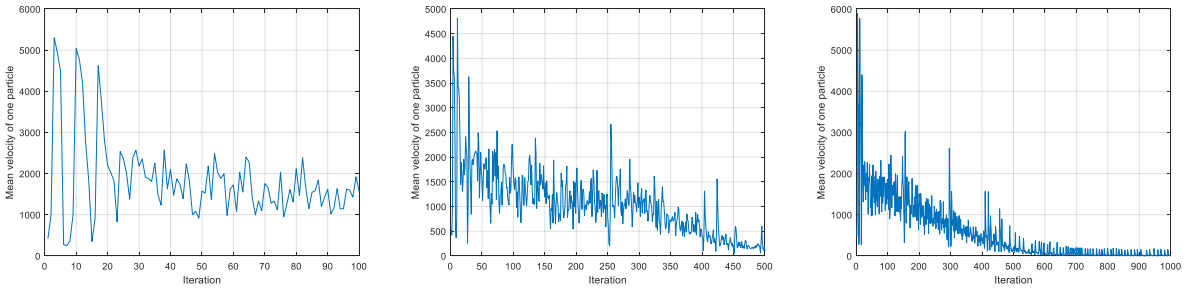
$$\begin{cases} a^{t+1} = ra^t \\ a^0 = Cste \end{cases} \quad (1.38)$$

Figure 1.5 shows an example (Schwefel's function) of the influence of the dynamic parameter strategy, where we can see clearly the damping of the jumps when the dynamic parameter is not used (Figure 1.5a) compared to the case when it is used (Figure 1.5b). In addition, a concept of iteration division, shown in Algorithm 1.2, is employed to reach a balanced diversification and intensification process. For the first number of iterations t_v, the range of velocity $[v_{min}, v_{max}]$ is selected using the equation (1.39) with $k^t = 1$ $(t < t_v)$ and for the rest of the iterations with $k^t \in (0,1)$ $(t > t_v)$. k is a constant called the decreasing velocity coefficient and belongs to the interval $(0,1)$ (Kassoul et al., 2021).

$$\begin{cases} v_{max}^t = k^t x_{max} \\ v_{min}^t = k^t x_{min} \end{cases} \quad (1.39)$$



(a) Without the dynamic parameter a^t



(b) With the dynamic parameter a^t

Figure 1.5. The Influence of the dynamic parameter strategy (Schwefel's function) (Kassoul et al., 2021)

A dynamic inertia weight strategy is integrated into the proposed algorithm, where the value of w is linearly decreased from w_{max} (initial value) to w_{min} (final value) according to equation (1.40), where r is the damping coefficient selected within the interval (0,1]. With this strategy, the tendency of particles to global search is decreased continuously, and at the same time, their tendency to local search increases continuously.

$$w^{t+1} = r\left(\frac{1-w^t}{1+w^t}\right) \tag{1.40}$$

Algorithm 1.2. Velocity controller

Set the value of t_v and k (t_v *is the exploration number of iterations,* k *is the decreasing velocity coefficient*)

- $v_{max} = x_{max}$
- $v_{min} = x_{min}$

While (number of iterations $\geq t_v$), **do**

- $v_{max} = kx_{max}$
- $v_{min} = kx_{min}$
- $w = r\left(\frac{1-w}{1+w}\right)$

end while

The pseudo-code of the strategy of the ExPSO method is described in Algorithm 1.3.

Algorithm 1.3. ExPSO

Input: optimization problem with n variables, a population size of N particles, $MaxFES$
Output: fitness value p_{gbest} of the encountered global best solution for the problem

Step1: Initialization

- Set the initial parameters $a, b, c, d, e, c_1, c_2, r, w, N, MaxFES$
- Initialize randomly and uniformly the position of the particles x_i and the velocity v_i $(i = 1,2, \dots, N)$.
- Determine the fitness value of particles and set the first best and worst position values p_{best_i} and p_{worst_i} equal to the actual positions of particles x_i $(i = 1,2, \dots, N)$
- Determine the global best and worst positions p_{gbest} and p_{gworst}

Step 2: Update of velocity and position of particles

While (the termination condition is not satisfied), **do**

for i = 1 to N

- Update the velocity of particles using Algorithm 1.1
- Apply velocity limits using the equations $v_{i,j} = \max(v_{i,j}; v_{min})$ and $v_i = \min(v_{i,j}; v_{max})$, where $v_{i,j}$ denotes the j^{th} velocity coordinate of the i^{th} particle
- Update the position of particles using the equation $x_i = x_i + v_i$
- Apply position limits using the equations $x_{i,j} = \max(x_{i,j}; l_b)$ and $x_i = \min(x_{i,j}; u_b)$, where $x_{i,j}$ denotes the j^{th} position coordinate of the i^{th} particle

Step 3: Update of personal and global best/worst positions

- Evaluate the fitness values of particles
- Update personal and global best positions of particles
- Update personal and global worst positions of particles
- Update the dynamic parameter a using the equation $a = r \cdot a$
- Decrease the velocity of particles using Algorithm 1.2

end while

1.4 Experimental results

To test the performance of the proposed ExPSO, comprehensive experiments are conducted on a set of 29 commonly used benchmark functions, including 7 unimodal functions, 6 multimodal functions, 10 fixed-dimensional multimodal functions, 6 composition functions (Table 1.2). The next section presents experiments on 4 classical engineering optimization problems (pressure vessel, compression spring, welded beam, and speed reducer designs). All experiments are conducted under the termination criterion used in the CEC 2020 competition (Suganthan, 2019). Thus, all the algorithms terminate when reaching $5\text{x}10^4$, $1\text{x}10^6$ and $2\text{x}10^7$ function evaluations (MaxFES) for problem with dimension D = 2~6, D = 10 and D = 30, respectively,

or when the error value is smaller than 10^{-8} (i.e., $\left|f(\vec{x}_{best}) - f(\overrightarrow{x^*})\right| \leq 10^{-8}$, where $\vec{x}_{best}$ is the best solution obtained so far by the algorithm and $\overrightarrow{x^*}$ is the global optimal solution). 30 independent runs are conducted for each function and each algorithm, and error values smaller than 10^{-8} will be taken as zero. The algorithms are implemented in Matlab R2020a and the experiments are performed on a PC with a 1.9-GHz Core (TM) i5CPU processor with 8 GB of RAM. The details on the experimental settings are provided below. The simulation results (including the performances of the standard PSO with different parameters), comparison with other benchmark algorithms (on the 29 benchmark functions and the engineering designs) are reported and discussed, and the statistical results and convergence performance are presented.

1.4.1 Experimental settings

Table 1.2 summarizes the 29 benchmark functions used for the first set of experiments, reporting the dimension of functions D, the range of the search space, the minimum value of the objective function f_{min}, and the description of functions. These functions are widely used in the literature for testing the performance of the algorithms (Das and Pratihar, 2021; Faramarzi et al., 2020; Heidari et al., 2019; Mirjalili and Lewis, 2016).The test functions include a set of 13 scalable problems of dimension D = 30, where the functions $f_1 - f_7$ are unimodal, $f_8 - f_{13}$ are multimodal, $f_{14} - f_{23}$ are non-scalable fixed-dimension (D = 2~6) multimodal functions, and 6 composite functions $CF1 - CF6$. The unimodal functions are suitable for exploitation search and the convergence of the algorithm because they have only one global optimum, whereas multimodal and composite functions have multiple local optima, and they are used for exploration search. Furthermore, composite test functions are more challenging because they increase the complexity of a real search space.

Table 1.2. Description of the 29 benchmark functions

	D	Search space	f_{min}	f_{min}	Name
Unimodal functions	30	[−100,100]	0	$f_1(x) = \sum_{i=1}^{n} x_i^2$	*Sphere Function*
		[−10,10]	0	$f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	*Schwefel's Problem 2.22*
		[−100,100]	0	$f_3(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2$	*Schwefel's Problem 1.2*
		[−100,100]	0	$f_4(x) = max\lvert x_i \rvert, 0 \leq i \leq n$	*Schwefel's Problem 2.21*
		[−30,30]	0	$f_5(x) = \sum_{i=1}^{n} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	*Rosenbrock Function*
		[−100,100]	0	$f_6(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$	*Step Function*
		[−1.28,1.28]	0	$f_7(x) = \sum_{i=1}^{n} i x_i^4$	*Noise Function 'Quartic'*
Multimodal	30	[−500,500]	−1.25e+04	$f_8(x) = 418.9829 \cdot n - \sum_{i=1}^{N} x_i sin\left(\sqrt{x_i}\right)$	*Schwefel's Function*
		[−5.12,5.12]	0	$f_9(x) = \sum_{i=1}^{n} (x_i^2 - 10cos(2\pi x_i) + 10)$	*Rastrigin's Function*

		[−32,32]	0	$f_{10}(x) = -20exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\right) - exp\left(\frac{1}{N}\sum_{i=1}^{N} cos2\pi x_i\right) + 20 + e$	*Ackley's Function*
		[−600,600]	0	$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	*Griewank's Function*
		[−5.12,5.12]	0	$f_{12}(x) = \frac{\pi}{n}\{10sin^2(\pi y_i) + \sum_{i=1}^{n-1}(y_i - 1)^2\,[1 + 10sin^2(\pi y_{i+1})] + (y_n - 1)^2\} + \sum_{i=1}^{n} u(x_i, 10,100,4)$ where $y_i = 1 + \frac{1}{4}(x_i + 1)$, $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i - a)^m, & x_i < a \end{cases}$	*Generalized Penalized Function 1*
		[−50,50]	0	f_{13}=$0.1\{sin^2(3\pi x_i) + \sum_{i=1}^{n}(x_i - 1)^2\,[1 + sin^2(3\pi x_i + 1)] + (x_n - 1)^2[1 + sin^2(2\pi x_i)]\} + \sum_{i=1}^{n} u(x_i, 5,100,4)$	*Generalized Penalized Function 2*
fixed−dimension multimodal functions	2	[−65.53,65.53]	0.998	$f_{14} = \left(\frac{1}{500} + \sum_{j=1}^{25}\frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right)^{-1}$	*Foxholes Function*
	4	[−5,5]	3.00e−04	f_{15}= $\sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$	*Kowalik function*
	2	[−5,5]	−1.0316	$f_{16} = 4x_1^2 + 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	*Six Hump Camel function*
	2	[−5,0] × [10,15]	0.397887	$f_{17} = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right) + 10\left(1 - \frac{1}{8\pi}\right)cos(x_1) + 10$	*Branin function*
	2	[−2,2]	3	$f_{18} = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \cdot [30 + (2x_1 - 3x_2)^2 \cdot (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	*Goldstein−Price function*
	3	[0,1]	−3.86	$f_{19} = -\sum_{i=1}^{4} c_i exp\left(-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right)$	*Hartman 3 function*
	6	[0,1]	−3.32	$f_{20} = -\sum_{i=1}^{4} c_i exp\left(-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right)$	*Hartman 6 function*
	4	[0,10]	−10.1532	$f_{21} = -\sum_{i=1}^{5}[(X - a_i)(X - a_i)^T + c_i]^{-1}$	*Shekel 5 function*
	4	[0,10]	−10.4029	$f_{22} = -\sum_{i=1}^{7}[(X - a_i)(X - a_i)^T + c_i]^{-1}$	*Shekel 7 function*
	4	[0,10]	−10.5363	$f_{23} = -\sum_{i=1}^{10}[(X - a_i)(X - a_i)^T + c_i]^{-1}$	*Shekel 10 function*
Composite functions	10	[−5,5]	0	$f_1, f_2, \ldots, f_{10}$= Sphere function $[\sigma_1, \sigma_2, \ldots, \sigma_{10}] = [1,1, \ldots,1]$ $[\lambda_1, \lambda_2, \ldots, \lambda_{10}] = [5/100,5/100, \ldots,5/100$	*Composition Function 1 (CF1)*
	10	[−5,5]	0	$f_1, f_2, \ldots, f_{10}$= Griewank's function $[\sigma_1, \sigma_2, \ldots, \sigma_{10}] = [1,1, \ldots,1]$ $[\lambda_1, \lambda_2, \ldots, \lambda_{10}] = [5/100,5/100, \ldots,5/100]$	*Composition Function 2 (CF2)*
	10	[−5,5]	0	$f_1, f_2, \ldots, f_{10}$= Griewank's function $[\sigma_1, \sigma_2, \ldots, \sigma_{10}] = [1,1, \ldots,1]$ $[\lambda_1, \lambda_2, \ldots, \lambda_{10}] = [1,1, \ldots,1]$	*Composition Function 3 (CF3)*
	10	[−5,5]	0	f_1, f_2= Ackley's function, f_3, f_4= Rastrigin's function, f_5, f_6=Weierstrass function, f_7, f_8= Griewank's function, f_9, f_{10}= Sphere function $[\sigma_1, \sigma_2, \ldots, \sigma_{10}] = [1,1, \ldots,1]$ $[\lambda_1, \lambda_2, \ldots, \lambda_{10}] = [5/32,5/32,1,1,5/0.5,5/0.5,$ $5/100,5/100,5/100,5/100]$	*Composition Function 4 (CF4)*
	10	[−5,5]	0	f_1, f_2= Rastrigin's function, f_3, f_4= Weierstrass function, f_5, f_6= Griewank's function, f_7, f_8= Ackley's function, f_9, f_{10}= Sphere function $[\sigma_1, \sigma_2, \ldots, \sigma_{10}] = [1,1, \ldots,1]$ $[\lambda_1, \lambda_2, \ldots, \lambda_{10}] = [1/5,1/5,5/0.5,5/0.5,5/100,$ $5/100,5/32,5/32,5/100,5/100]$	*Composition Function 5 (CF5)*

10	[−5,5]	0	f_1, f_2= Rastrigin's function, f_3, f_4= Weierstrass function, f_5, f_6= Griewank's function, f_7, f_8= Ackley's function, f_9, f_{10}= Sphere function $[\sigma_1, \sigma_2, \dots, \sigma_{10}] = [0.1,0.2,0.3,0.4,0.5, 0.6,0.7,0.8,0.9,1]$ $[\lambda_1, \lambda_2, \dots, \lambda_{10}] = [0.1 * 1/5,0.2 * 1/5, 0.3 * 5/0.5,0.4 * 5/0.5,0.5 * 5/100,0.6 * 5/100, 0.7 * 5/32,0.8 * 5/32,0.9 * 5/100,1 * 5/100]$	*Composition Function 6 (CF6)*

1.4.2 Results on benchmark functions

1.4.2.1 Standard PSO with different parameters

In this subsection, we investigate the effect of new values of the cognitive and social scaling parameters c_1 and c_2, and we compare the obtained results with the standard PSO (Shi and Eberhart, 1998). Two cases are considered, named $\mathrm{C_N}$PSO and $\mathrm{S_N}$PSO. $\mathrm{C_N}$PSO has a Negative Cognitive parameter $c_1 = -1$, and a social parameter $c_2 = 2$. $\mathrm{S_N}$PSO has a cognitive parameter $c_1 = 2$ and a Negative Social parameter $c_2 = -1$. The idea of choosing negative values for the parameters c_1 and c_2 is, because particles of the basic PSO often converge prematurely towards local optima, to investigate opposite regions to those followed by the particles for personal or global best positions. Therefore, $\mathrm{C_N}$PSO is the case when particles follow the global best position and move against the personal best positions, whereas $\mathrm{S_N}$PSO is the case when particles follow the personal best positions and move against the global best position. This could allow escaping from any eventual local minimum and exploring new regions. Table 1.3 reports the average value and the standard deviation of the results obtained by PSO, $\mathrm{C_N}$PSO and $\mathrm{S_N}$PSO on the 23-benchmark functions over 30 independent runs. For this experiment, the population size N and the maximum of function evaluations *MaxFES* are set to 30 and 15000, respectively.

The results of the best algorithm are marked in bold. The results show the superiority of $\mathrm{C_N}$PSO and $\mathrm{S_N}$PSO over PSO. This is, certainly, because particles can explore new regions instead of always moving directly to the probable global and best positions. We note that these results have contributed to the integration of personal and global worse positions in the proposed algorithm.

Table 1.3. Results on the 23 classical benchmark functions for different values of acceleration' coefficients

f	f_{min}	**PSO** $c_1 = c_2 = 2$		**C_NPSO** $c_1 = -1$, $c_2 = 2$		**S_NPSO** $c_1 = 2$, $c_2 = -1$	
		Average	Standard deviation	Average	Standard deviation	Average	Standard deviation
f_1	0	4.11e+01	3.73e+01	**0.00e+00**	0.00e+00	**0.00e+00**	0.00e+00
f_2	0	1.25e+00	1.24e+00	**0.00e+00**	0.00e+00	**0.00e+00**	0.00e+00
f_3	0	5.50e+02	3.87e+02	**0.00e+00**	0.00e+00	**0.00e+00**	0.00e+00
f_4	0	4.69e+00	2.79e+00	**0.00e+00**	0.00e+00	**0.00e+00**	0.00e+00
f_5	0	8.60e+02	1.06e+03	**2.90e+01**	1.61e−02	**2.90e+01**	1.48e−02
f_6	0	5.25e+01	5.52e+02	**0.00e+00**	0.00e+00	**0.00e+00**	0.00e+00
f_7	0	1.75e−04	2.16e−04	**0.00e+00**	0.00e+00	**0.00e+00**	0.00e+00
f_8	−1.25e+04	−5.63e+03	8.83e+02	**−6.68e+03**	8.75e+02	−6.48e+03	9.15e+02
f_9	0	8.14e+00	9.78e+00	**0.00e+00**	0.00e+00	**0.00e+00**	0.00e+00
f_{10}	0	2.97e+00	1.90e+00	**0.00e+00**	0.00e+00	**0.00e+00**	0.00e+00
f_{11}	0	1.01e+00	6.02e−01	**0.00e+00**	0.00e+00	**0.00e+00**	0.00e+00
f_{12}	0	1.49e+00	1.53e+00	8.56e−01	2.35e−01	**8.08e−01**	3.34e−01
f_{13}	0	7.80e+00	8.53e+00	**2.79e+00**	2.22e−01	**2.79e+00**	1.73e−01
f_{14}	0.998	**9.98e−01**	1.33e−05	1.01e+00	6.03e−02	1.40e+00	8.79e−01
f_{15}	3.00e−04	**6.97e−04**	2.81e−04	2.02e−03	7.91e−04	3.59e−03	5.06e−03
f_{16}	−1.0316	**−1.03e+00**	5.54e−15	**−1.03e+00**	2.93e−03	**−1.03e+00**	3.20e−03
f_{17}	0.397887	5.32e−01	2.27e−01	**3.98e−01**	7.46e−04	**3.98e−01**	1.01e−03
f_{18}	3	**3.00e+00**	2.69e−13	**3.00e+00**	2.05e−03	**3.00e+00**	4.09e−03
f_{19}	−3.86	−3.35e+00	2.98e−01	**−3.84e+00**	2.60e−02	−3.75e+00	1.03e−01
f_{20}	−3.32	−1.51e+00	4.60e−01	−3.12e+00	7.67e−02	−3.05e+00	7.67e−02
f_{21}	−10.1532	−5.04e−01	1.97e−01	−9.39e+00	1.09e+00	**−9.69e+00**	6.19e−01
f_{22}	−10.4029	−9.16e+00	6.72e+01	**−9.89e+00**	5.95e−01	−9.77e+00	6.52e−01
f_{23}	−10.5363	−9.10e−01	3.01e−01	**−9.64e+00**	1.30e+00	−9.44e+00	1.38e+00

1.4.2.2 ExPSO on benchmark problems

To assess the performance of ExPSO and validate its effectiveness, it is tested for 23 benchmark functions and 6 composite functions (conceived in the CEC 2005 special session (Liang et al., 2005)) and compared to 11 state-of-the-art algorithms: Phasor Particle Swarm Optimization (PPSO) (Ghasemi et al., 2019), Fitness-based Multi-role PSO (FMPSO) (Xia, Xing, et al., 2019), Multi-Swarm PSO (MSPSO) (Xia, Gui, and Zhan, 2018), eXpanded PSO (XPSO) (Xia et al., 2020), Triple Archives PSO (TAPSO) (Xia, Gui, et al., 2019), Equilibrium Optimizer (EO) (Faramarzi et al., 2020), fitness-Distance balance-based Symbiotic Organism Search (DSOS) (Kahraman et al., 2020), Harris Hawks Optimization (HHO) (Heidari et al., 2019), Improved Real-coded Genetic Algorithm (IRGA) (Das and Pratihar, 2021), the winner of CEC 2017 (Effective Butterfly Optimizer with Covariance Matrix Adapted Retreat Phase (EBOwithCMAR) (Kumar et al., 2017)), and the winner of CEC 2020 (Improved Multi-

Operator DE (IMODE) (Sallam et al., 2020)). The average value (Avg.) and the standard deviation (S.D.) are reported for each experiment. The results of the best algorithm are marked in bold. The parameters settings for ExPSO are detailed in Table 1.4, those of the compared algorithms in Table 1.5, and the comparison results are reported in Table 1.6.

Table 1.4. Summary of settings and research process of the proposed method

Parameter Settings		**Research process**
Exponential weight parameter	$a = 2$	Update at each iteration t
Cognitive acceleration best-coefficient	$b = 2$	▪ *Update of velocity*
Social acceleration best-coefficient	$c = 2$	• For the N_1 particles of subpopulation 1
Cognitive acceleration worst-coefficient	$d = -1$	$v_i = wv_i + a\gamma R_{vec} + br_1(p_{best_i} - x_i) + cr_2(p_{gbest} - x_i) + dr_3(p_{worst_i} - x_i) + er_4(p_{gworst} - x_i)$
Social acceleration worst-coefficient	$e = -1$	• For the N_2 particles of subpopulation 2
Cognitive scaling parameter	$c_1 = -1$	$v_i = wv_i + a\gamma R_{vec} + br_1(p_{best_i} - x_i) + cr_2(p_{gbest} - x_i)$
Social scaling parameter	$c_2 = 2$	• For the rest of the swarm particles
Inertia weight	$w = 0.9$	$v_i = wv_i + c_1 r_1(p_{best_i} - x_i) + c_2 r_2(p_{gbest} - x_i)$
Damping coefficient	$r = 0.9$	▪ *Dynamic inertia weight strategy*
Exploration number of iterations	$t_v = 10$	$w = r\left(\frac{1-w}{1+w}\right)$
Decreasing velocity coefficient	$k = 0.2$	▪ *Linear decreasing exponential weight parameter*
Number of particles of subpopulation 1	$N_1 = N/3 = 10$	$a = ra$
Number of particles of subpopulation 2	$N_2 = N/3 = 10$	Update after t_v iterations
Number of particles of subpopulation 3	$N_3 = N/3 = 10$	▪ *Decreasing velocity*
		$v_{max} = kx_{max}$; $v_{min} = kx_{min}$

Table 1.5. Brief description of parameters settings of the compared algorithms

Algorithm	**Parameters**
PPSO	Inertia weight $w = 0$, control all parameters by periodic functions Cos and Sin
FMPSO	$c_h = 1.2, c_m = 0.7, c_l = 0.2, \| s_{dh} \| = 0.6 * D,$ $\| s_{dm} \| = 0.3 * D, \| s_{dl} \| = 0.1 * D$
MSPSO	$\chi = 0.7298$, $c_1 = c_2 = 1.49445$, $R_n = 10$
XPSO	$\eta = 0.2$, $Stag_{max} = 5$, $p = 0.5$
TAPSO	$w = 0.7298$, $p_c = 0.5$, $p_m = 0.02$, $R_n = 10$, M= N/4
EO	$a_1 = 2, a_2 = 1$ and generation probability $GP = 0.5$
DSOS	weight coefficient $w \in [0.4, 0.6]$
HHO	Initial escaping energy $E_0 \in [-1, 1]$
IRGA	Crossover, mutation, variable wise crossover, and directional probabilities $p_c = 0.98$, $p_m = 0.03, p_{cv} = 0.94, p_d = 0.5$, and multiplying factor $\alpha = 0.95$
EBOwithCMAR	$PS_{1,max} = 18D, PS_{1,min} = 4, PS_{2,max} = 46.8D, PS_{2,min} = 10,$ $PS_3 = 4 + (3log(D)[20])$ $CS = 100{,}200$ and 300, H = 6[8], $\sigma = 0.3$,
IMODE	$prob_{ls} = 0.1, cfe_{ls} = 0.25 * \text{MaxFES}, NP^{init} = 6 * D^2, NP^{min} = 4, A = 2.6,$ $H = 20 * D, FES_{LS} = 0.85 * MaxFES$

The test on unimodal functions $f_1 - f_7$ is suitable to investigate the exploitation ability of the algorithms. It can be seen from Table 1.6 that all algorithms provide the global optimum for all unimodal functions $f_1 - f_7$, except FMPSO for f_5, XPSO for $f_1, f_3 - f_5$ and TAPSO for f_5, f_7. It should be noted that ExPSO has achieved the global minimum in each run over the 30 runs for functions $f_1 - f_5, f_7$. According to the results of ExPSO in dealing with these unimodal

functions, we note that the exponential PSO strategy provides a significant improvement of the exploitation potential.

For multimodal functions $f_8 - f_{13}$, ExPSO, DSOS, HHO and IRGA can locate the global minimum for all functions, whereas PPSO and IMODE (resp. TAPSO) has not achieved the global minimum only for function f_8 (resp. f_{11}). Hence, ExPSO can be considered, in terms of exploration ability, as a reliable and better optimizer. For the fixed-dimension multimodal functions $f_{14} - f_{23}$, the results clearly show that ExPSO achieves the global optimum for all problems except f_{20}, where it obtains a slightly lower value to the minimum value. We note that in these test functions, EBOwithCMAR provides a better exploration ability and finds the optimum in all problems, PPSO has not achieved the global minimum for functions $f_{15} - f_{20}$, whereas the other algorithms have achieved the global minimum only for function $f_{14}, f_{16} - f_{19}$. The results for these functions indicate that ExPSO has also a very good ability to explore the search space more deeply.

The tests on composition functions help to evaluate the quality of balancing the exploration and exploitation search process. It covers six composite functions by rotating, expanding, shifting, and hybridizing multimodal and unimodal functions. Each composition function is given by:

$$F(x) = \sum_{i=1}^{n} \{w_i * [f_i'(x - o_{inew} + o_{iold}/\lambda_i * M_i) + bias_i]\} \quad (1.41)$$

where n is the number of basic functions, $f_i'(x)$ is the i^{th} normalized basic function, w_i is a weight value, M_i is the orthogonal rotation matrix, o_{inew} is the new shifted optimum position, o_{iold} is the old optimum position for the i^{th} basic function, λ_i is a factor used to stretch or compress the i^{th} basic function, and $bias_i$ is used to place anywhere the global optimum of the i^{th} basic function. These functions are described in detail in (Liang et al., 2005). From Table 1.6 ($bias_i \neq 0$), we observe that EBOwithCMAR provides the best value for three functions $CF_3 - CF_5$ and IMODE (resp. DSOS) achieves the better result for CF_1, CF_2 (resp. CF_6). The performance of the proposed algorithm in this category of test functions (i.e., composition functions) is lower compared to previous tests (i.e., unimodal, multimodal, and fixed-multimodal functions). An adjustment of the ExPSO parameters may help to address this weakness. Although ExPSO is not ranked first in this test category, the results show that for most functions, it is very competitive with the other best algorithms.

To allow attempting niching algorithms to find all the global optima in one run, a good test on these composition functions will be by setting $bias_i$ to 0 for all f_i (Liang et al., 2005). The results, reported in Table 1.5 ($bias_i = 0$), show that in all these hybrid composite test functions, ExPSO and HHO are better than the other algorithms and obtain the global minimum in all test functions.

Table 1.6. Comparison results for the 29 benchmark functions

Unimodal functions								
Method	**Result**	f_1	f_2	f_3	f_4	f_5	f_6	f_7
ExPSO	Avg.	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
	S.D.	0.00e+00	0.00e+00	0.00e+00	0.00e+00	3.17e−09	0.00e+00	0.00e+00
PPSO	Avg.	**0.00e+00**	**0.00e+00**	**0.00e+00**	7.99e−05	**0.00e+00**	**0.00e+00**	**0.00e+00**
	S.D.	5.54e−10	1.70e−09	1.25e−10	1.52e−04	1.18e−09	0.00e+00	3.41e−10
FMPSO	Avg.	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	7.97e−01	**0.00e+00**	**0.00e+00**
	S.D.	2.43e−09	6.30e−10	4.55e−10	1.50e−10	1.62e+00	0.00e+00	7.26e−10
MSPSO	Avg.	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
	S.D.	1.54e−09	6.02e−10	3.42e−10	4.44e−10	2.60e−09	0.00e+00	1.89e−10
XPSO	Avg.	2.64e−05	**0.00e+00**	8.33e+00	1.81e−01	9.25e+00	**0.00e+00**	**0.00e+00**
	S.D.	1.24e−04	5.65e−10	3.31e−04	2.05e−01	9.16e+00	0.00e+00	1.79e−09
TAPSO	Avg.	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
	S.D.	7.66e−10	7.96e−10	2.27e−10	1.61e−10	1.05e−10	0.00e+00	1.68e−10
EO	Avg.	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	7.17e−03	**0.00e+00**	9.22e−07
	S.D.	1.85e−09	1.07e−09	1.59e−09	8.81e−10	3.87e−02	7.74e−10	4.37e−07
DSOS	Avg.	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
	S.D.	2.07e−09	1.68e−09	1.60e−09	1.01e−09	4.63e−10	0.00e+00	2.27e−09
HHO	Avg.	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
	S.D.	2.93e−09	2.73e−09	2.93e−09	6.64e−09	2.82e−09	2.33e−10	6.86e−09
IRGA	Avg.	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	1.31e−07
	S.D.	2.74e−09	3.78e−10	1.68e−11	1.56e−10	4.02e−12	0.00e+00	5.38e−10
EBOwithCMAR	Avg.	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
	S.D.	5.61e−10	4.28e−10	1.48e−10	1.24e−10	2.20e−10	0.00e+00	9.19e−10
IMODE	Avg.	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
	S.D.	1.33e−09	8.94e−10	3.05e−09	2.50e−09	8.31e−10	0.00e+00	2.43e−09
Multimodal functions								
Method	**Result**	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	
ExPSO	Avg.	**−1.25e+04**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	
	S.D.	1.91e−01	0.00e+00	0.00e+00	0.00e+00	2.73e−09	3.21e−09	
PPSO	Avg.	−1.19e+04	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	
	S.D.	7.26e+02	1.96e−10	2.80e−10	2.24e−10	3.07e−09	1.27e−10	
FMPSO	Avg.	−1.10e+04	2.28e+01	**0.00e+00**	5.34e−02	1.38e−02	2.20e−03	
	S.D.	4.22e+02	1.64e+01	3.66e−10	4.58e−02	5.92e−02	4.47e−03	
MSPSO	Avg.	**−1.25e+04**	**0.00e+00**	1.09e−08	2.62e−03	**0.00e+00**	**0.00e+00**	
	S.D.	4.07e−01	3.46e−09	6.27e−09	5.13e−03	2.16e−09	3.20e−09	
XPSO	Avg.	−1.08e+04	3.98e+00	**0.00e+00**	3.53e−02	3.44e−08	**0.00e+00**	
	S.D.	3.19e+02	6.12e+02	6.12e−10	3.41e−02	1.36e−07	7.51e−10	
TAPSO	Avg.	**−1.25e+04**	**0.00e+00**	**0.00e+00**	7.78e−03	**0.00e+00**	**0.00e+00**	
	S.D.	2.51e−01	3.00e−09	5.45e−10	1.58e−02	2.29e−09	2.93e−09	
EO	Avg.	−9.11e+03	**0.00e+00**	**0.00e+00**	**0.00e+00**	6.91e−03	8.01e−03	
	S.D.	6.83e+02	1.44e−09	1.22e−09	1.55e−09	2.63e−02	3.58e−02	
DSOS	Avg.	**−1.25e+04**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	
	S.D.	3.26e−01	1.44e−09	1.59e−09	2.06e−09	1.31e−09	2.76e−09	
HHO	Avg.	**−1.25e+04**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	
	S.D.	3.31e−01	3.06e−09	2.85e−09	2.78e−09	3.42e−09	2.51e−09	
IRGA	Avg.	**−1.25e+04**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	
	S.D.	2.55e−01	5.48e−09	9.65e−09	1.83e−09	7.50e−02	7.70e−10	
EBOwithCMAR	Avg.	−7.18e+03	**0.00e+00**	**0.00e+00**	5.75e−04	2.76e−02	8.00e−03	
	S.D.	5.32e+02	1.36e−09	3.85e−10	2.21e−03	6.05e−02	1.62e−02	
IMODE	Avg.	−7.19e+03	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	
	S.D.	6.37e+02	2.73e−09	9.26e−10	2.52e−09	3.93e−09	1.72e−09	

Table 1.6. (Continued)

Method	Result	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}	f_{22}	f_{23}
ExPSO	Avg.	**9.98e−01**	**3.07e−04**	**−1.03e+00**	**3.97e−01**	**3.00e+00**	**−3.86e+00**	−3.30e+00	**−1.01e+01**	**−1.04e+01**	**−1.05e+01**
	S.D.	3.06e−05	7.53e−07	3.21e−04	2.88e−05	2.64e−04	2.82e−04	5.63e−02	2.70e−04	2.62e−04	2.55e−04
PPSO	Avg.	**9.98e−01**	5.69e−03	**−1.03e+00**	**3.97e−01**	**3.00e+00**	**−3.86e+00**	−3.28e+00	**−1.01e+01**	**−1.04e+01**	**−1.05e+01**
	S.D.	2.44e−04	9.00e−03	2.84e−04	2.94e−05	2.62e−04	2.78e−04	5.68e−02	1.37e+00	3.14e−04	1.41e+00
FMPSO	Avg.	**9.98e−01**	3.07e−03	**−1.03e+00**	**3.97e−01**	**3.00e+00**	**−3.86e+00**	−3.25e+00	−5.58e+00	−7.15e+00	−9.08e+00
	S.D.	2.79e−04	6.90e−03	3.17e−04	3.19e−05	2.72e−04	2.63e−04	5.90e−02	2.37e+00	3.21e+00	2.77e+00
MSPSO	Avg.	**9.98e−01**	4.13e−04	**−1.03e+00**	**3.97e−01**	**3.00e+00**	**−3.86e+00**	−3.22e+00	−7.91e+00	−9.20e+00	−8.98e+00
	S.D.	3.38e−04	3.27e−04	2.88e−04	3.08e−05	2.75e−04	2.71e−05	9.90e−05	3.31e+00	2.75e+00	2.89e+00
XPSO	Avg.	**9.98e−01**	3.94e−04	**−1.03e+00**	**3.97e−01**	**3.00e+00**	**−3.86e+00**	−3.30e+00	−7.23e+00	−8.95e+00	−9.77e+00
	S.D.	2.65e−04	2.48e−04	3.48e−04	2.84e−05	2.91e−04	2.29e−04	4.82e−02	3.48e+00	2.97e+00	2.34e+00
TAPSO	Avg.	**9.98e−01**	1.01e−03	**−1.03e+00**	**3.97e−01**	**3.00e+00**	**−3.86e+00**	−3.27e+00	−5.89e+00	−6.25e+00	−5.17e+00
	S.D.	3.00e−04	3.66e−03	2.95e−04	3.05e−05	2.72e−04	2.53e−04	6.01e−02	1.16e+00	3.14e+00	3.37e+00
EO	Avg.	**9.98e−01**	1.73e−03	**−1.03e+00**	**3.97e−01**	**3.00e+00**	**−3.86e+00**	−3.27e+00	−9.31e+00	−1.02e+01	−9.81e+01
	S.D.	2.53e−04	5.07e−03	2.79e−04	2.77e−05	3.05e−04	2.30e−04	6.69e−02	1.93e+00	9.70e−01	9.87e−01
DSOS	Avg.	**9.98e−01**	**3.07e−04**	**−1.03e+00**	**3.97e−01**	**3.00e+00**	**−3.86e+00**	−3.29e+00	**−1.01e+01**	**−1.04e+01**	**−1.05e+01**
	S.D.	2.37e−04	3.16e−08	2.72e−04	2.48e−05	2.32e−04	3.47e−04	5.33e−02	2.66e−04	2.66e−04	3.05e−04
HHO	Avg.	**9.98e−01**	3.20e−04	**−1.03e+00**	**3.97e−01**	**3.00e+00**	**−3.86e+00**	−3.20e+00	−5.78e+00	−5.44e+00	−5.12e+00
	S.D.	3.32e−04	1.25e−05	2.74e−04	2.49e−05	2.38e−04	1.29e−03	6.64e−02	9.29e−01	1.33e+00	1.39e+00
IRGA	Avg.	**9.98e−01**	1.38e−03	**−1.03e+00**	**3.97e−01**	**3.00e+00**	**−3.86e+00**	−3.29e+00	−8.15e+00	−9.55e+00	−9.59e+00
	S.D.	2.61e−04	3.59e−03	3.03e−04	2.74e−05	2.68e−04	2.26e−04	5.52e−02	3.37e+00	2.40e+00	2.73e+00
EBOwithCMAR	Avg.	**9.98e−01**	**3.07e−04**	**−1.03e+00**	**3.97e−01**	**3.00e+00**	**−3.86e+00**	**−3.32e+00**	**−1.01e+01**	**−1.04e+01**	**−1.05e+01**
	S.D.	3.39e−16	1.77e−19	5.19e−06	0.00e+00	2.68e−09	2.71e−15	1.29e−15	7.17e−15	3.30e−16	4.30e−15
IMODE	Avg.	**9.98e−01**	**3.07e−04**	**−1.03e+00**	**3.97e−01**	**3.00e+00**	**−3.86e+00**	**−3.32e+00**	**−1.01e+01**	**−1.04e+01**	**−1.05e+01**
	S.D.	3.39e−16	1.85e−19	6.76e−16	0.00e+00	2.99e−09	2.71e−15	1.36e−15	7.06e−15	0.00e+00	5.00e−15

Composite functions

Method	Result	*CF*1		*CF*2		*CF*3		*CF*4		*CF*5		*CF*6	
		$Bias \neq 0$	$Bias = 0$	$Bias \neq 0$	$Bias = 0$	$Bias \neq 0$	$Bias = 0$	$Bias \neq 0$	$Bias = 0$	$Bias \neq 0$	$Bias = 0$	$Bias \neq 0$	$Bias = 0$
ExPSO	Avg.	5.33e+01	**0.00e+00**	8.43e+01	**0.00e+00**	2.11e+02	**0.00e+00**	3.98e+02	**0.00e+00**	4.41e+01	**0.00e+00**	6.96e+02	**0.00e+00**
	S.D.	7.76e+01	0.00e+00	6.51e+01	0.00e+00	7.84e+01	0.00e+00	1.32e+02	0.00e+00	7.53e+01	0.00e+00	1.96e+02	0.00e+00
PPSO	Avg.	7.00e+01	**0.00e+00**	1.03e+02	3.53e+00	2.58e+02	5.09e+00	4.36e+02	2.14e+00	9.17e+01	8.71e+00	6.82e+02	1.42e+00
	S.D.	8.37e+01	1.36e−09	1.08e+02	1.13e+01	1.04e+02	2.78e+01	9.40e+02	3.91e+01	1.02e+01	2.62e+01	2.05e+02	1.25e+00
FMPSO	Avg.	4.33e+01	**0.00e+00**	4.26e+01	8.69e+00	1.57e+02	8.94e+01	4.23e+02	8.10e+00	4.69e+01	4.00e+01	7.36e+02	2.92e+00
	S.D.	7.28e+01	1.27e−09	7.76e+01	6.08e+00	6.17e+01	3.87e+01	3.09e+02	2.49e+01	1.01e+02	5.63e+01	1.24e+02	1.40e+00
MSPSO	Avg.	2.33e+01	**0.00e+00**	6.76e+01	4.90e+00	1.47e+02	9.24e+01	4.13e+02	**0.00e+00**	3.07e+01	2.22e+01	7.28e+02	1.58e+00
	S.D.	4.30e+09	1.45e−09	6.66e+01	6.35e+00	4.95e+02	5.20e+01	3.29e+02	1.97e−09	5.38e+01	4.40e+01	1.92e+02	7.51e−01
XPSO	Avg.	5.33e+01	**0.00e+00**	1.47e+02	1.57e+00	2.00e+02	3.61e+01	4.25e+02	8.87e−01	4.80e+01	8.58e+01	8.22e+02	2.50e+00
	S.D.	8.60e+01	1.27e−09	1.11e+02	5.01e+01	1.19e+01	4.64e+01	1.60e+02	2.40e+00	9.71e+01	5.90e+01	1.63e+02	8.13e−00
TAPSO	Avg.	7.00e+01	**0.00e+00**	1.41e+02	3.25e+00	1.77e+02	1.60e+02	3.48e+02	8.90e+00	5.36e+01	6.77e+01	7.55e+02	2.80e+00
	S.D.	9.15e+01	1.74e−09	8.75e+01	5.57e+00	6.40e+01	8.04e+01	1.06e+02	4.79e+01	5.88e+01	6.21e+01	1.97e+02	2.15e+00
EO	Avg.	7.66e+01	**0.00e+00**	6.57e+01	6.49e−01	1.48e+02	6.04e+01	3.40e+02	**0.00e+00**	4.54e+01	5.14e+01	8.22e+02	2.39e+00
	S.D.	8.17e+01	1.62e−09	8.07e+01	1.53e+00	3.35e+01	4.85e+01	1.09e+02	1.60e−09	8.13e+01	6.22e+01	1.66e+02	9.33e−00
DSOS	Avg.	5.00e+01	**0.00e+00**	3.97e+01	**0.00e+00**	1.61e+02	9.98e+00	3.66e+02	**0.00e+00**	3.85e+01	1.61e+01	**4.13e+02**	2.07e+00
	S.D.	6.30e+01	2.54e−09	4.91e+01	3.64e−09	4.83e+01	2.44e+01	1.93e+02	3.03e−09	4.77e+01	1.54e+01	1.07e+02	1.07e+00
HHO	Avg.	1.24e+02	**0.00e+00**	1.89e+02	**0.00e+00**	3.91e+02	**0.00e+00**	4.22e+02	**0.00e+00**	9.29e+01	**0.00e+00**	6.64e+02	**0.00e+00**
	S.D.	1.16e+02	3.10e−09	2.19e+02	3.23e−09	1.99e+01	2.74e−09	2.51e+02	3.12e−09	7.20e+01	3.37e−09	1.96e+02	3.19e−09
IRGA	Avg.	3.00e+01	**0.00e+00**	8.89e+01	5.75e+00	1.70e+02	1.44e+02	3.63e+02	1.31e+00	5.37e+01	5.92e+01	7.61e+02	2.47e+00
	S.D.	5.96e+01	1.40e−09	8.88e+01	7.31e+00	5.43e+01	6.01e+01	1.16e+02	2.71e+00	7.18e+01	6.59e+01	1.90e+02	1.06e−01
EBOwithCMAR	Avg.	1.00e+01	**0.00e+00**	4.76e+01	**0.00e+00**	**4.50e+01**	2.07e+01	**2.49e+02**	**0.00e+00**	**2.02e+00**	3.92e+00	5.24e+02	2.35e+00
	S.D.	3.05e+01	1.20e−09	6.25e+01	1.48e−09	5.38e+01	2.53e+01	7.18e+01	1.49e−09	4.06e+00	1.72e+01	1.04e+02	4.65e−04
IMODE	Avg.	**0.00e+00**	**0.00e+00**	**9.69e+00**	**0.00e+00**	1.15e+02	1.93e+00	3.13e+02	**0.00e+00**	3.31e+00	**0.00e+00**	5.27e+02	8.64e−01

In summary, the results reported in Table 1.6 show that ExPSO exhibits excellent performance on unimodal, multimodal and fixed-dimensional multimodal functions. In the case of composition function, the performance of ExPSO was very competitive compared to the other best algorithm in this category.

1.4.3 Convergence analysis

The convergence history allows the observation of the evolution of the solution fitness during the search process. Therefore, the convergence curves of ExPSO and the eleven competing algorithms for unimodal functions $f_1 - f_7$ are plotted in Figure 1.6. ExPSO provides a high-speed convergence for all functions $f_1 - f_7$. For f_2, f_3, f_4 and f_5, the fast convergence is more

visible. It can be seen that DSOS and HHO present almost the same convergence rate as ExPSO for these functions. We note that FMPSO and IMODE need more time to converge, and all algorithms tend to converge towards the optimum in less than 200 iterations for f_1, f_5, f_6 and f_7.

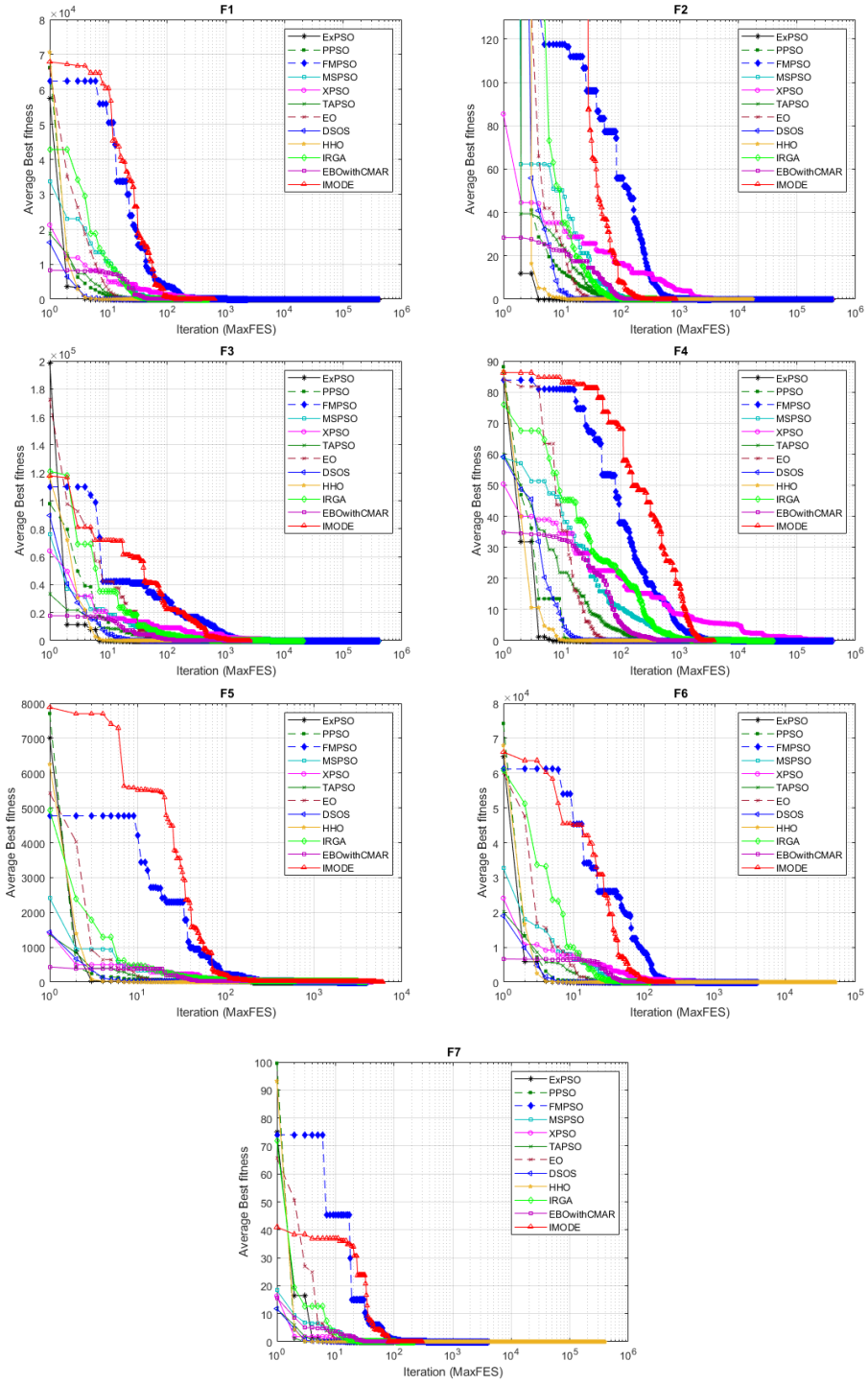


Figure 1.6. Convergence curves of ExPSO for unimodal functions

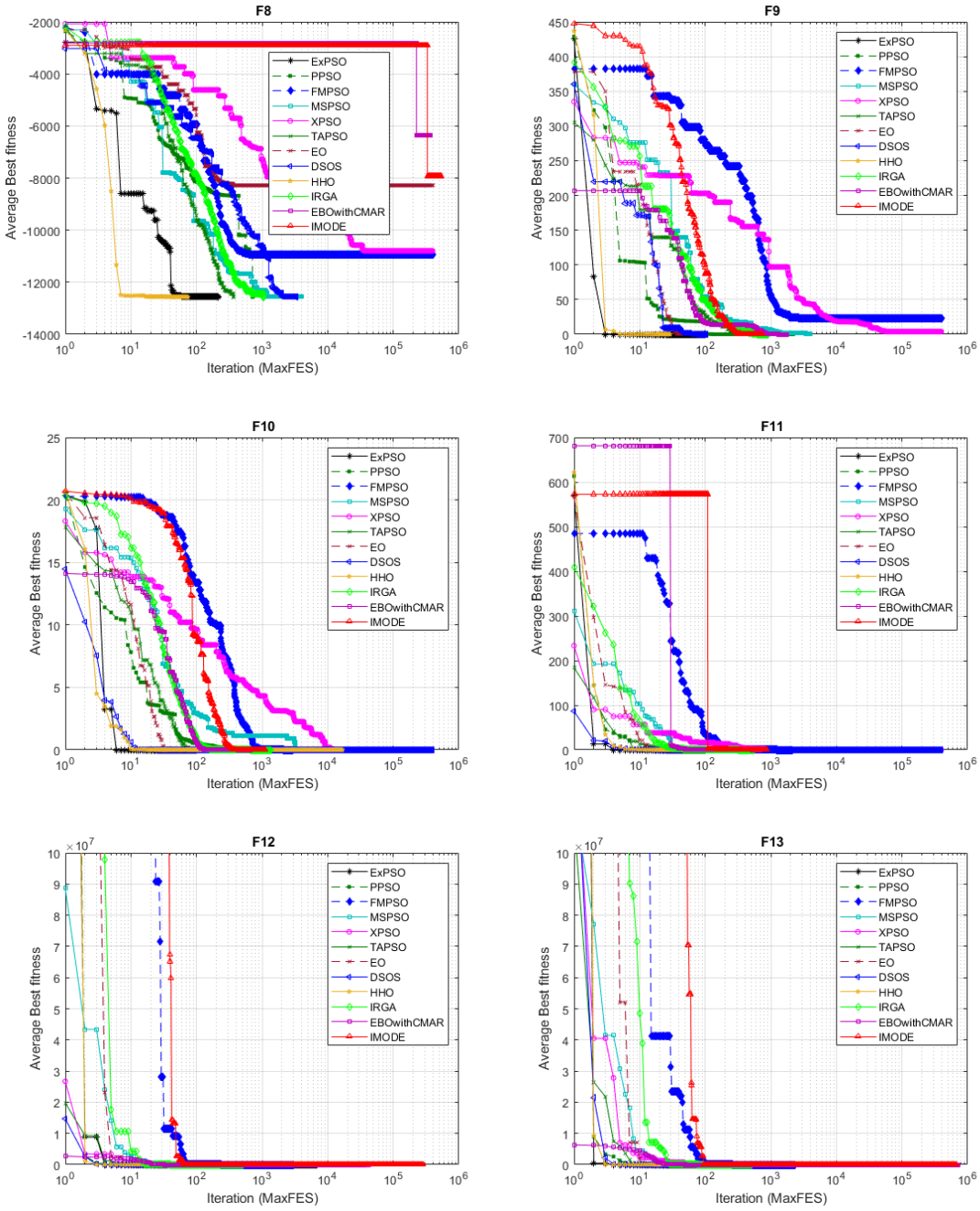


Figure 1.7. Convergence curves of ExPSO for multimodal functions

From the curves on multimodal functions (Figure 1.7), it can be seen that ExPSO and HHO achieve a higher-speed convergence compared to the other algorithms for f_9 and f_{10}. DSOS presents the same convergence rate for f_{10}. HHO converges faster than ExPSO for f_8, whereas EO, EBOwithCMAR and IMODE are easily trapped in local optimums at different levels, and FMPSO and XPSO converge prematurely into local minima.

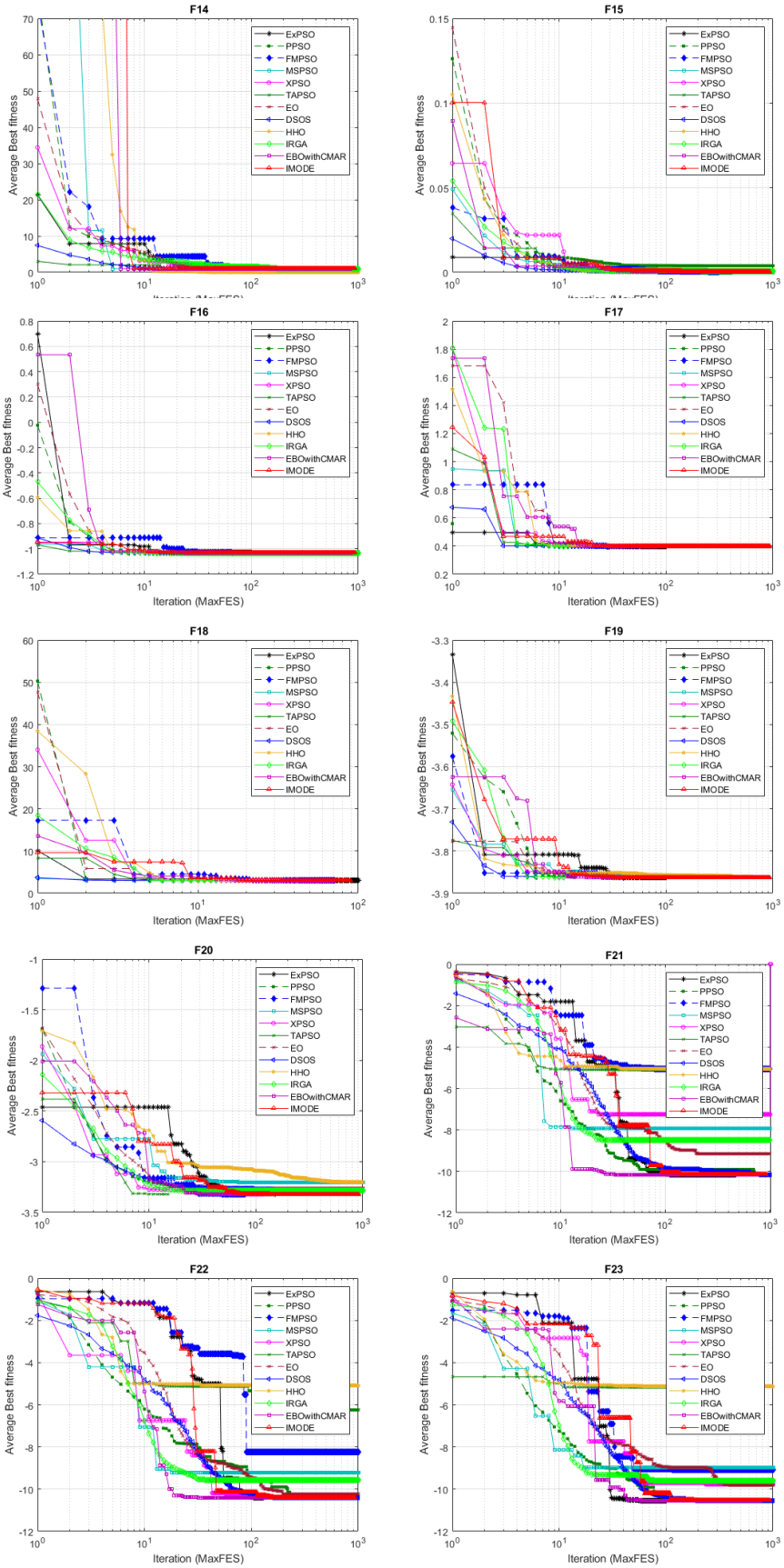


Figure 1.8. Convergence curves of ExPSO for fixed–dimension multimodal functions

Figure 1.8 provides the convergence curves for the fixed-dimension multimodal functions $f_{14} - f_{23}$. ExPSO provides better convergence for all functions. For f_{20}, XPSO, TAPSO and IRGA are the most successful. We note that ExPSO, PPSO, DSOS, EBOwithCMAR and IMODE converge to the global optimum for $f_{21} - f_{23}$, whereas the other algorithms are trapped in different local minima at different levels.

To summarize, convergence graphs show that ExPSO exhibits high-speed convergence in most benchmark problems. These results show that ExPSO has high stability for different problem types and can effectively balance local and global search abilities in the search process.

1.4.4 Algorithm Complexity

The computational complexity of ExPSO is calculated as in the competition CEC 2020 for dimensions D in {5, 10, 15}. The time durations of T_0, T_1 and T_2 are in seconds (Suganthan, 2022). Table 1.7 presents a summary of the obtained results.

Table 1.7. Algorithm Complexity

D	T_0	T_1	$\widehat{T_2}$	$\frac{\widehat{T_2} - T_1}{T_0}$
5		0.5370	5.6343	111.17
10	0.045849	0.5605	5.7949	114.16
15		0.5957	6.0717	119.43

1.4.5 Statistical analysis

The non-parametric Wilcoxon signed test with a 5% significance level) is considered to test the validity of the results, judge which algorithms are better/worse than ExPSO, and measure whether the improvements achieved by ExPSO are meaningful. Table 1.8 presents the p-value and z-value of ExPSO versus other competitor optimizers. Since the degree of significance is 5%, the p-values greater than 0.05 indicate that the difference in the results between the two methods is not significant. Only three *p*-value marked in bold (i.e., 0.654159 for ExPSO vs. DSOS, 0.616450 for ExPSO vs. EBOwithCMAR and 0.582920 for ExPSO vs. IMODE) are greater than 0.05, proving that ExPSO is significantly better than the other techniques in almost all cases and that it significantly outperforms the other algorithms for the different benchmark problems.

Considering the Friedman mean rank, ExPSO is ranked first among the approaches considering all types of functions. The Friedman test is conducted with 5% significance.

According to the obtained p-value of 1.3113e−13 (Table 1.7), it is noticed that the results of ExPSO are significantly better in almost all cases than those achieved by other methods. This means that ExPSO is powerful and has a proper balance ability to exploit and explore the search space more deeply

Table 1.8. Friedmann mean rank, p-values, and z-values of the Wilcoxon signed test with 5% significance

Algorithm	**Rank**	**Friedman mean rank**	**p-value** (ExPSO vs.)	**z-value** (ExPSO vs.)
ExPSO	**1**	4.44	-	-
PPSO	6	6.64	0.000068	-3.984589
FMPSO	11	8.38	0.004682	-2.828147
MSPSO	5	6.58	0.014889	-2.435076
XPSO	12	8.55	0.000044	-4.084250
TAPSO	9	7.24	0.000477	-3.493144
EO	7	6.96	0.000314	-3.603880
DSOS	4	5.42	**0.654159**	-0.447992
HHO	10	7.32	0.000040	-4.106905
IRGA	8	6.99	0.001731	-3.132935
EBOwithCMAR	3	4.97	**0.616450**	-0.500888
IMODE	2	4.50	**0.582920**	-0.549125
	p-value	1.3113 e−13	**Wilcoxon signed test**	

1.5 Results on engineering problems

In this section, ExPSO is tested on four well-known engineering design problems: Pressure Vessel Design (PVD), Compression Spring Design (CSD), Welded Beam Design (WBD), and Speed Reducer Design (SRD). These problems have different constraints which should be not violated by the optimal solution(s) obtained, and thus a constraint handling method must be utilized. The penalty function, a standard, and simple constraint handling method (Coello and Montes, 2002) (which assigns a large penalty coefficient) is adopted. So, the unconstrained state of the constrained design is expressed by the fitness function $f_{fit}(\vec{x})$ as follows:

$$f_{fit}(\vec{x}) = f_{obj}(\vec{x}) + f_{cons}(\vec{x}) \tag{1.42}$$

where $f_{obj}(\vec{x})$ is the objective function, and the violation of constraints is represented by the penalty term $f_{cons}(\vec{x})$ and is formulated as follows:

$$f_{cons}(\vec{x}) = P_1 \sum_{i=1}^{n} \max\left(0, g_i(\vec{x})\right) + P_2 \sum_{j=1}^{m} \max\left(0, \left|h_j(\vec{x})\right| - \delta\right) \tag{1.43}$$

where $g_i(\vec{x})$ and $h_j(\vec{x})$ are, respectively, the i^{th} inequality and j^{th} equality constraints, δ is any very small number and P_1 and P_2 are penalty coefficients that should be large enough to adequately penalize the objective function under inequality/equality constraints.

We note that for the engineering problems presented in this study, all different constraints are inequality constraints, and thus $h(\vec{x}) = 0$ or $m = 0$. For a fair comparison, the same penalty function is used for all methods. In all the experiments, the population size and the maximum number of iterations are set to 30 and 500, respectively. The results of ExPSO for the four engineering problems are summarized in Table 1.9.

Table 1.9. Summary of results of ExPSO for the four engineering problems

Pressure vessel design			
Decision variable		Constraint	
T_s	0.8125	$g_1(X)$	-4.15e-09
T_h	0.4375	$g_2(X)$	-0.03588
R	42.09845	$g_3(X)$	0
L	176.6366	$g_4(X)$	-63.3634
$f(x)$		**6059.714**	

Compression Spring Design			
Decision variable		Constraint	
d	0.051489	$g_1(X)$	-1.26e-14
D	0.351929	$g_2(X)$	0
N	11.75319	$g_3(X)$	-4.0442
		$g_4(X)$	-0.73105
$f(x)$		**0.01265665**	

Welded Beam Design			
Decision variable		Constraint	
h	0.20571	$g_1(X)$	-1.09801
l	3.2540	$g_2(X)$	-1.50108
t	9.0372	$g_3(X)$	-2.35e-01
b	0.20572	$g_4(X)$	-7.05e-05
		$g_5(X)$	-6.51968
		$g_6(X)$	-8.07e-02
		$g_7(X)$	-3.40970
$f(x)$		**1.695329**	

Speed reducer design			
Decision variable		Constraint	
b	3.5	$g_1(X)$	-2.17713
m	0.7	$g_2(X)$	-9.85e+01
z	17	$g_3(X)$	-1.92584
l_1	7.3	$g_4(X)$	-1.82817
l_2	7.71551	$g_5(X)$	-3.63435
d_1	3.35055	$g_6(X)$	0
d_2	5.28668	$g_7(X)$	-2.80e+01
		$g_8(X)$	-1.32e-03
		$g_9(X)$	-6.99867
		$g_{10}(X)$	-3.74e-01
		$g_{11}(X)$	-4.25e-03
$f(x)$		2994.424	

1.5.1 Pressure Vessel Design

The first considered benchmark problem is the PVD (Sandgren, 1990) illustrated in Figure 1.9. The pressure vessel is a cylindrical vessel capped ends at both ends by hemispherical heads. The PVD has a minimum volume of $21.24\,m^3$ and should work under the pressure of $2.2 \cdot 10^7\ Pa\ (3000\ psi)$. The objective is to minimize the total fabrication cost, including the cost of material, forming, and welding. There are four mixed design variables $(x_1 - x_4)$; two discrete variables $(x_1 - x_2)$ taken as a multiple of 0.0625 (shell thickness $T_s(= x_1)$, head thickness $T_h(= x_2)$), and two continuous variables (inner radius $R(= x_3)$, length of the cylindrical section without the head $L(= x_4)$).

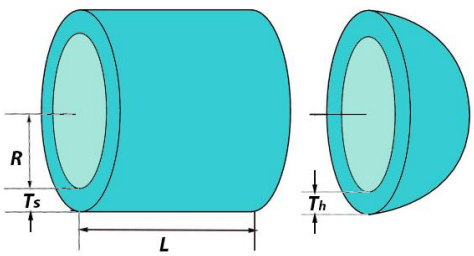


Figure 1.9. Pressure vessel design (Das and Pratihar, 2021)

The problem and its constraints can be given as follows:

Consider $\vec{x} = [x_1\ x_2\ x_3\ x_4] = [T_s\ T_h R\ L]$

Minimize $f_{cost}(\vec{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3$

Subject to

$$g_1(\vec{x}) = -x_1 + 0.0193x_3 \le 0$$
$$g_2(\vec{x}) = -x_2 + 0.00954x_3 \le 0$$
$$g_3(\vec{x}) = -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \le 0 \quad (1.44)$$
$$g_4(\vec{x}) = x_4 - 240 \le 0$$

Variables range

$$0 \le x_1 \le 99$$
$$0 \le x_2 \le 99$$
$$10 \le x_3 \le 200$$
$$10 \le x_4 \le 200$$

Table 1.10 gives the results obtained by ExPSO and the other methods. We can see that ExPSO, EO, and DSOS generated almost the same minimum cost.

Table 1.10 Comparison of optimum results and statistical results for the PVD problem

f	ExPSO	PPSO	FMPSO	MSPSO	XPSO	TAPSO	EO	DSOS	HHO	IRGA	EBO/ CMAR	IMODE
T_s	0.8125	0.8125	1.0625	0.8125	4.6875	0.9375	0.8125	0.8125	0.875	0.875	0.875	0.8125
T_h	0.4375	0.5	0.5625	0.4375	1.8125	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375
R	42.0984	42.1752	55.0518	45.3367	143.44	45.8595	42.0984	42.0984	42.0572	45.0935	45.0464	41.3216
L	176.636	176.643	62.7144	140.253	43.9896	135.007	176.636	176.636	177.147	142.749	143.237	186.506
$f(x)$	**6059.71**	6216.25	6771.59	6090.52	44232.7	6424.06	**6059.71**	**6059.71**	6064.73	6118.47	6123.95	6156.61
Best	6059.71	6216.25	6771.59	6090.52	44232.7	6424.06	6059.71	6059.71	6064.73	6118.47	6123.95	6156.61
Worst	7332.84	13846.4	27782.6	7461.07	99145.7	6424.06	7544.49	6820.41	7544.49	7544.92	7473.84	20218.5
Mean	6197.53	6473.90	17380.0	6547.37	70587.6	6424.06	6641.71	6095.44	6684.65	6863.61	6979.85	9355.38
S.D.	330.552	1852.43	10507.0	791.289	27522.7	0.00000	566.007	148.222	425.166	372.505	399.293	3079.24

1.5.2 Compression spring design

In this classical mechanical engineering problem (Coello and Montes, 2002), the objective is to minimize the tension spring weight (see Figure 1.10) under four constraints on minimum deflection, and limits on the outside diameter shear surge frequency stress. The diameter of the wire $d(= x_1)$, the diameter of coil $D(= x_2)$ and the number of active coils $N(= x_3)$ are treated as the design variables.

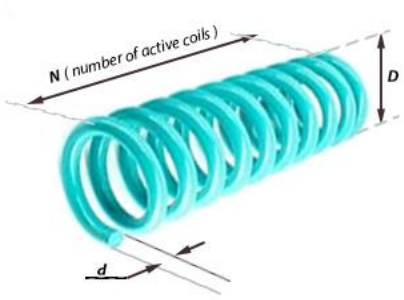


Figure 1.10. Compression Spring Design (Das and Pratihar, 2021)

The mathematical formulation is formulated as follows:

Consider $\vec{x} = [x_1\ x_2\ x_3] = [d\ D\ N]$

Minimize $f_{cost}(\vec{x}) = (x_3 + 2)x_2x_1^2$

Subject to

$$g_1(\vec{x}) = 1 - \frac{x_2^3 x_3}{71785x_1^4} \leq 0$$

$$g_2(\vec{x}) = \frac{4x_2^2 - x_1x_2}{12566(x_2x_1^3 - x_1^4)} + \frac{1}{5108x_1^2} - 1 \leq 0 \quad (1.45)$$

$$g_3(\vec{x}) = 1 - \frac{140.45x_1}{x_2^2x_3} \leq 0$$

$$g_4(\vec{x}) = \frac{x_1 + x_2}{1.5} - 1 \leq 0$$

Variables range

$$0.05 \leq x_1 \leq 2$$

$$0.25 \leq x_2 \leq 1.3$$

$$2.00 \leq x_3 \leq 15$$

Table 1.11. Comparison of optimum results and statistical results for the CSD problem

f	ExPSO	PPSO	FMPSO	MSPSO	XPSO	TAPSO	EO	DSOS	HHO	IRGA	EBO/ CMAR	IMODE
d	0.05148	0.05213	0.05	0.05	0.36478	0.8125	0.05282	0.05190	0.05162	0.05343	0.05190	0.05167
D	0.35192	0.36750	0.31742	0.31742	0.51232	0.5	0.38472	0.36202	0.35507	0.40020	0.36185	0.35640
N	11.7531	10.6834	14.0278	14.0278	12.2503	42.0979	9.81786	10.9849	11.3858	9.13057	10.9940	11.3075
$f(x)$	**0.01266**	0.01267	0.01272	0.01272	0.97149	0.01279	0.01225	**0.01266**	**0.01266**	0.01272	**0.01266**	**0.01266**
Best	0.01266	0.01266	0.01271	0.01271	0.97149	0.01279	0.01225	0.01266	0.01266	0.01271	0.01266	0.01266
Worst	0.01523	0.01777	5.4901	0.20984	2.0172	0.01279	0.01331	0.01268	0.01727	0.01665	0.01297	0.01300
Mean	0.01285	0.01327	2.3035	0.09407	1.3531	0.01279	0.01286	0.01267	0.13643	0.01368	0.01271	0.01269
S.D.	0.00052	0.00127	2.8465	0.10297	0.5773	0.00000	0.00017	0.00000	0.00100	0.00091	0.00000	0.00000

Table 1.11 provides the best solution and statistical results realized by ExPSO and the other algorithms. It can be seen that all the algorithms obtain almost the same result except XPSO. We note that ExPSO, DSOS, HHO, EBOwithCMAR and IMODE obtain the same results.

1.5.3 Welded beam design

The welded beam design was introduced by Coello (2000). Another well-known engineering optimization problem (Figure 1.11) consists of determining four variables (i.e., the thickness of weld $h(=x_1)$, the length of the clamped bar $l(=x_2)$, the height of the bar $t(=x_3)$, and the thickness of the bar $b(=x_4)$) in order to minimize the fabrication cost of the welded beam. The design must satisfy seven constraints such as shear stress (τ), bending stress (σ), buckling load (P_c), deflection (δ), and three other constraints.

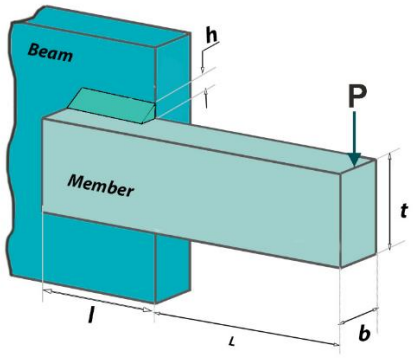


Figure 1.11. Welded beam design (Das and Pratihar, 2021)

The mathematical formulation is expressed as follows:

Consider $\vec{x} = [x_1\ x_2\ x_3\ x_4] = [h\ l\ t\ b]$

Minimize $f_{cost}(\vec{x}) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2)$

Subject to

$$\begin{aligned} &g_1(\vec{x}) = \tau(\vec{x}) - \tau_{max} \le 0 \\ &g_2(\vec{x}) = \sigma(\vec{x}) - \sigma_{max} \le 0 \\ &g_3(\vec{x}) = \delta(\vec{x}) - \delta_{max} \le 0 \\ &g_4(\vec{x}) = x_1 - x_4 \le 0 \\ &g_5(\vec{x}) = P - P_c(\vec{x}) \le 0 \\ &g_6(\vec{x}) = 0.125 - x_1 \le 0 \\ &g_7(\vec{x}) = 0.11047x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5.0 \le 0 \end{aligned} \tag{1.46}$$

Variables range

$$\begin{aligned} &0.1 \le x_1 \le 2 \\ &0.1 \le x_2 \le 10 \\ &0.1 \le x_3 \le 10 \\ &0.1 \le x_4 \le 2 \end{aligned}$$

where

$$\tau(\vec{x}) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2}$$

$$\tau' = \frac{P}{\sqrt{2}x_1x_2},\ \tau = \frac{MR}{J}$$

$$M = P\left(L + \frac{x_2}{2}\right), R = \sqrt{\frac{x_2^2}{4} + (\frac{x_1+x_3}{2})^2}$$

$$J = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{4} + \left(\frac{x_1+x_3}{2}\right)^2\right]\right\}$$

$$\sigma(\vec{x}) = \frac{6PL}{x_4x_3^2},\ \delta(\vec{x}) = \frac{4PL^3}{Ex_3^3x_4}$$

$$P_c(\vec{x}) = \frac{4.013E\sqrt{\frac{x_3^2x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right)$$

$$P = 6{,}000\ lb,\ L = 14\ in,\ \delta_{max} = 0.25\ in$$

$$E = 30 \times 10^6\ psi,\ G = 12 \times 10^6\ psi$$

$$\tau_{max} = 13{,}600\ psi,\ \sigma_{max} = 30{,}000\ psi$$

The results are summarized in Table 1.12. ExPSO exhibits almost the same result as EO, DSOS, EBOwithCMAR and IMODE, and it outperforms all the other algorithms. Further, the analysis of the statistical results shows that ExPSO has the mean solution close to the best one. This shows that ExPSO has good stability during all the runs.

Table 1.12. Comparison of optimum results and statistical results for the WBD problem

f	ExPSO	PPSO	FMPSO	MSPSO	XPSO	TAPSO	EO	DSOS	HHO	IRGA	EBO/ CMAR	IMODE
h	0.20571	0.20558	0.20575	0.19479	0.57852	0.20577	0.20572	0.8125	0.20170	0.18613	0.20573	0.20575
l	3.2540	3.255	3.2529	3.4454	1.4751	3.2526	3.2531	0.4375	3.3639	3.6459	3.2531	3.2529
t	9.0372	9.0393	9.0362	9.0366	3.9777	9.0357	9.0366	8.9921	45.0464	9.0171	9.0366	9.0361
b	0.20572	0.20572	0.20575	0.20573	0.44424	0.20577	0.20572	0.20776	143.237	0.20661	0.20573	0.20575
$f(x)$	**1.69532**	1.69564	1.69746	1.70960	1.86153	1.69540	**1.69524**	**1.69524**	1.71191	1.72122	**1.69525**	**1.69538**
Best	1.69532	1.69564	1.69746	1.70960	1.86153	1.69540	1.69524	1.69524	1.71191	1.72122	1.69525	1.69534
Worst	1.84125	1.79378	2.00492	1.95573	24.1573	1.69540	1.69719	1.69524	2.24232	3.23536	1.69525	2.12349
Mean	1.70795	1.7438	1.79853	1.81655	9.45822	1.69540	1.69539	1.69524	1.81616	2.20120	1.69525	1.83024
S.D.	0.02952	0.04909	0.17875	0.12616	6.68331	0.00000	0.00046	0.00000	0.10774	0.36434	0.00000	0.11752

1.5.4 Speed reducer design

The SRD (Speed Reducer Design, see Figure 1.12) is one of the essential parts of the gearbox. It was first studied by Golinski (1970). The objective is to minimize the weight of the speed reducer subject to 11 constraints. The design of the speed reducer is considered with seven design variables: face width $b(= x_1)$, teeth module $m(= x_2)$, number of pinion teeth $N(= x_3)$,

shaft length 1 $l_1(=x_4)$, shaft length 2 $l_2(=x_5)$, shaft diameter 1 $d_1(=x_6)$, and shaft diameter 2 $d_2(=x_7)$.

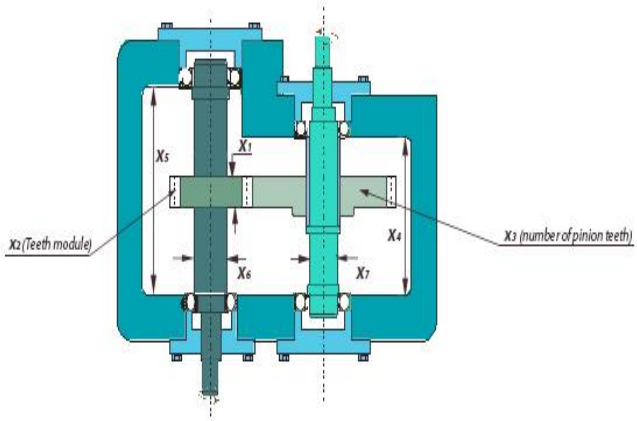


Figure 1.12. Speed reducer design (Bayzidi et al., 2021)

Mathematically the problem can be stated as follows:

Consider $\vec{x} = [x_1\ x_2\ x_3\ x_4 x_5\ x_6\ x_7] = [b\ m\ N\ l_1\ l_2\ d_1\ d_2]$

Minimize $f_{cost}(X) = 0.7854x_1x_2^2(3.3333x_3^2 + 14.9334x_3 - 43.0934) - 1.508(x_6^2 + x_7^2) + 7.4777(x_6^3 + x_7^3) + 0.7854(x_4x_6^2 + x_5x_7^2)$

Subject to

$$g_1(\vec{x}) = \frac{27}{x_1x_2^2x_3} - 1 \le 0$$

$$g_2(\vec{x}) = \frac{397.5}{x_1x_2^2x_3^2} - 1 \le 0$$

$$g_3(\vec{x}) = \frac{1.93x_4^3}{x_2x_6^4x_3} - 1 \le 0 \qquad (1.47)$$

$$g_4(\vec{x}) = \frac{1.93x_5^3}{x_2x_7^4x_3} - 1 \le 0$$

$$g_5(\vec{x}) = \frac{\sqrt{(745x_4/x_2x_3)^2 + 16.9\cdot10^6}}{110x_6^3} - 1 \le 0$$

$$g_6(\vec{x}) = \frac{\sqrt{(745x_5/x_2x_3)^2 + 157.5\cdot10^6}}{85x_7^3} - 1 \le 0$$

$$g_7(\vec{x}) = \frac{x_2x_3}{40} - 1 \le 0$$

$$g_8(\vec{x}) = \frac{5x_2}{x_1} - 1 \le 0$$

$$g_9(\vec{x}) = \frac{x_1}{12x_2} - 1 \le 0$$

$$g_{10}(\vec{x}) = \frac{1.5x_6 + 1.9}{x_4} - 1 \le 0$$

$$g_{11}(\vec{x}) = \frac{1.1x_7 + 1.9}{x_5} - 1 \le 0$$

Variables range

$$2.6 \leq x_1 \leq 3.6$$
$$0.7 \leq x_2 \leq 0.8$$
$$17 \leq x_3 \leq 28$$
$$7.3 \leq x_4 \leq 8.3$$
$$7.3 \leq x_5 \leq 8.3$$
$$2.9 \leq x_6 \leq 3.9$$
$$5 \leq x_7 \leq 5.5$$

Table 1.13 presents the comparison of the best optimal solution and the statistical results of ExPSO with the other optimization methods. Among the compared optimization algorithms, EBOwithCMAR obtains the best solution with a minimum of 2639.499 and all the other algorithms obtain the same value of 2994.424 except FMPSO (resp. XPSO and HHO) that finds a value of 3048.377 (resp. 3060.097 and 2997.639).

Table 1.13. Comparison of optimal results and statistical results for the SRD problem

f	ExPSO	PPSO	FMPSO	MSPSO	XPSO	TAPSO	EO	DSOS	HHO	IRGA	EBO/ CMAR	IMODE
b	3.5	3.5	3.503	3.50002	3.15449	3.5	3.5	3.5	3.50011	3.5	5.84706	3.5
m	0.7	0.7	0.7	0.7	0.70968	0.7	0.7	0.7	0.7	0.7	0.54743	0.7
z	17	17	17.232	17	17.5324	17	17	17	17	17	15.5055	17
l_1	7.3	7.3	7.96966	7.3	7.5323	7.3	7.3	7.3	7.3	7.3	7.29966	7.3
l_2	7.71532	7.71532	7.71532	7.72209	7.69776	7.71532	7.71532	7.71532	7.83025	7.71532	7.71684	7.71532
d_1	3.35054	3.35054	3.36859	3.35007	3.59568	3.35054	3.35054	3.35054	3.35299	3.35054	3.35968	3.35054
d_2	5.28665	5.28665	5.28881	5.28696	5.29491	5.28665	5.28665	5.28665	5.28669	5.28665	5.28794	5.28665
$f(x)$	2994.42	2994.42	3048.37	2994.65	3060.09	2994.42	2994.42	2994.42	2997.63	2994.42	**2639.49**	2994.42
Best	2994.40	2994.42	3048.37	2994.65	3060.09	2994.42	2994.42	2994.42	2997.63	2994.42	2639.49	2994.42
Worst	3033.74	3.05599	3285.24	3058.60	5383.88	2994.42	2994.42	2994.42	3803.21	2994.42	2992.38	2994.42
Mean	2999.97	3019.27	3156.64	3020.69	4153.01	2994.42	2994.42	2994.42	3114.53	2994.42	2848.01	2994.42
S.D.	9.18043	3.24611	119.737	33.5916	676.145	0.00000	0.00000	0.00000	180.949	0.00000	59.7684	0.00000

1.6 Conclusion

A metaheuristic should not only be assessed according to its quality (measured by the values of the obtained solutions), but also according to speed criteria (time required to generate competitive solutions), robustness (the standard deviation of the results being an indicator for it), and flexibility (ability to organize the method so that it can tackle different problems and objective functions). It is challenging to design a metaheuristic that performs well concerning various criteria. However, based on the existing literature, it is fair to say that methods combining efficiently diversification and intensification mechanisms are the most promising.

Due to the insufficient balance between exploration and exploitation and the premature convergence disadvantage of the standard PSO, a novel PSO variant called Exponential Particle Swarm Optimization (ExPSO) is proposed in this chapter. The main idea is to escape from local minima traps by exploring new regions of the solution space during the search. For this purpose, a leaping strategy is incorporated in the equation of the velocity using an exponential function allowing particles to make leaps and thus skip a possible local optimum and allowing particles to visit other regions in the search space. In addition, the population of particles is divided into subpopulations to maintain diversity in the particles of the swarm. To enhance a trade-off between the diversity of the population and the intensification process, a concept of iteration division is employed and a new linear decreasing cognitive parameter (with a dynamic inertia weight strategy) is integrated into the ExPSO algorithm.

Also, the effect of new values of cognitive and social scaling parameters of the standard PSO is investigated. The comparison with other well-known algorithms on a set of 29 well-known benchmark functions provides results that are generally significantly better than those obtained by state-of-the-art methods. The results obtained for the unimodal functions show the excellent exploitation ability of ExPSO, those obtained for the multimodal functions confirm its exploration ability, and the results for the composition functions demonstrate that ExPSO improves both the diversification and intensification ability during the search process. Finally, the results for four constrained engineering problems indicate that the performance of ExPSO is promising since it achieves highly competitive results in terms of solution quality compared to other optimizers. Furthermore, the results of the convergence evaluation and statistical analysis show that ExPSO performs better and faster.

Various research directions could be derived from this work. The first direction is to extend ExPSO to solve multi-objective problems. In addition, the proposed algorithm can be hybridized with other metaheuristics to further improve its performance. Finally, in line with the features proposed in (Schindl and Zufferey, 2015), a refined learning mechanism could be incorporated.

1.7 References

Alam, M. S., and Arefifar, S. A. (2020). Hybrid PSO-TS Based Distribution System Expansion Planning for System Performance Improvement Considering Energy Management. *IEEE Access*, *8*, 221599-221611.

Ardizzon, G., Cavazzini, G., and Pavesi, G. (2015). Adaptive acceleration coefficients for a new search diversification strategy in particle swarm optimization algorithms. *Information Sciences*, *299*, 337-378.

Bayzidi, H., Talatahari, S., Saraee, M., and Lamarche, C.-P. (2021). Social Network Search for Solving Engineering Optimization Problems. *Computational Intelligence and Neuroscience*, *2021*, e8548639.

Benvidi, A., Abbasi, S., Gharaghani, S., Tezerjani, M. D., and Masoum, S. (2017). Spectrophotometric determination of synthetic colorants using PSO–GA-ANN. *Food chemistry*, *220*, 377-384.

Chi, R., Su, Y., Zhang, D., Chi, X., and Zhang, H. (2019). A hybridization of cuckoo search and particle swarm optimization for solving optimization problems. *Neural Computing and Applications*, *31*(1), 653-670.

Clerc, M. (1999). The swarm and the queen : Towards a deterministic and adaptive particle swarm optimization. *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, *3*, 1951-1957.

Clerc, M., and Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE transactions on Evolutionary Computation*, *6*(1), 58-73.

Coello, C. A. (2000). Use of a self-adaptive penalty approach for engineering optimization problems. *Computers in Industry*, *41*(2), 113-127.

Coello, C. A., and Mezura-Montes, E. (2002). Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatics*, *16*(3), 193-203.

Dabhi, D., and Pandya, K. (2020). Enhanced Velocity Differential Evolutionary Particle Swarm Optimization for Optimal Scheduling of a Distributed Energy Resources With Uncertain Scenarios. *IEEE Access*, *8*,

Das, A. K., and Pratihar, D. K. (2021). Solving engineering optimization problems using an improved real-coded genetic algorithm (IRGA) with directional mutation and crossover. *Soft Computing*, *25*(7), 5455-5481.

El-Shafiey, M. G., Hagag, A., El-Dahshan, E.-S. A., and Ismail, M. A. (2022). A hybrid GA and PSO optimized approach for heart-disease prediction based on random forest. *Multimedia Tools and Applications*, *81*(13), 18155-18179.

Faramarzi, A., Heidarinejad, M., Stephens, B., and Mirjalili, S. (2020). Equilibrium optimizer : A novel optimization algorithm. *Knowledge-Based Systems*, *191*, 105190.

Fielding, B., and Zhang, L. (2020). Evolving deep denseBlock architecture ensembles for image classification. *Electronics*, *9*(11), 1880.

Gandomi, A. H., Yun, G. J., Yang, X.-S., and Talatahari, S. (2013). Chaos-enhanced accelerated particle swarm optimization. *Communications in Nonlinear Science and Numerical Simulation*, *18*(2), 327-340.

Garg, H. (2016). A hybrid PSO-GA algorithm for constrained optimization problems. *Applied Mathematics and Computation*, *274*, 292-305.

Ghasemi, M., Akbari, E., Rahimnejad, A., Razavi, S. E., Ghavidel, S., and Li, L. (2019). Phasor particle swarm optimization : A simple and efficient variant of PSO. *Soft Computing*, *23*(19), 9701-9718.

Ghorpade, S. N., Zennaro, M., Chaudhari, B. S., Saeed, R. A., Alhumyani, H., and Abdel-Khalek, S. (2021). Enhanced differential crossover and quantum particle swarm optimization for IoT applications. *IEEE Access*, *9*, 93831-93846.

Heidari, A. A., Mirjalili, S., Faris, H., Aljarah, I., Mafarja, M., and Chen, H. (2019). Harris hawks optimization : Algorithm and applications. *Future generation computer systems*, *97*, 849-872.

Ho, S.-Y., Lin, H.-S., Liauh, W.-H., and Ho, S.-J. (2008). OPSO : Orthogonal Particle Swarm Optimization and Its Application to Task Assignment Problems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, *38*(2), 288-298.

Issa, M., Helmi, A. M., Elsheikh, A. H., and Abd Elaziz, M. (2022). A biological sub-sequences detection using integrated BA-PSO based on infection propagation mechanism : Case study COVID-19. *Expert Systems with Applications*, *189*, 116063.

Jan Golinski. (1970). Optimal synthesis problems solved by means of nonlinear programming and random methods. *Journal of Mechanisms*, *5*(3), 287-309.

Javidrad, F., and Nazari, M. (2017). A new hybrid particle swarm and simulated annealing stochastic optimization method. *Applied Soft Computing*, *60*, 634-654.

Kahraman, H. T., Aras, S., and Gedikli, E. (2020). Fitness-distance balance (FDB) : A new selection method for meta-heuristic search algorithms. *Knowledge-Based Systems*, *190*, 105169.

Kassoul, K., Belhaouari, S. B., and Cheikhrouhou, N. (2021). Dynamic Cognitive-Social Particle Swarm Optimization. *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*, 200-205.

Kennedy, J., and Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95-international conference on neural networks*, *4*, 1942-1948.

Kennedy, J., and Mendes, R. (2002). Population structure and particle swarm performance. *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, *2*, 1671-1676.

Kumar, A., Misra, R. K., and Singh, D. (2017). Improving the local search capability of effective butterfly optimizer using covariance matrix adapted retreat phase. *2017 IEEE congress on evolutionary computation (CEC)*, 1835-1842.

Li, C., You, F., Yao, T., Wang, J., Shi, W., Peng, J., and He, S. (2021). Simulated annealing particle swarm optimization for high-efficiency power amplifier design. *IEEE Transactions on Microwave Theory and Techniques*, *69*(5), 2494-2505.

Li, Z., Wang, W., Yan, Y., and Li, Z. (2015). PS–ABC : A hybrid algorithm based on particle swarm and artificial bee colony for high-dimensional optimization problems. *Expert Systems with Applications*, *42*(22), 8881-8895.

Liang, J. J., Suganthan, P. N., and Deb, K. (2005). Novel composition test functions for numerical global optimization. *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, 68-75.

Lynn, N., Ali, M. Z., and Suganthan, P. N. (2018). Population topologies for particle swarm optimization and differential evolution. *Swarm and evolutionary computation*, *39*, 24-35.

Mahi, M., Baykan, Ö. K., and Kodaz, H. (2015). A new hybrid method based on Particle Swarm Optimization, Ant Colony Optimization and 3-Opt algorithms for Traveling Salesman Problem. *Applied Soft Computing*, *30*, 484-490.

Mashwani, W. K., Shah, S. N. A., Belhaouari, S. B., and Hamdi, A. (2020). Ameliorated Ensemble Strategy-Based Evolutionary Algorithm with Dynamic Resources Allocations. *International Journal of Computational Intelligence Systems.*

Mendes, R., Kennedy, J., and Neves, J. (2004). The fully informed particle swarm : Simpler, maybe better. *IEEE transactions on evolutionary computation*, *8*(3), 204-210.

Mirjalili, S., and Lewis, A. (2016). The Whale Optimization Algorithm. *Advances in Engineering Software*, *95*, 51-67.

Mirjalili, S., Mirjalili, S. M., and Lewis, A. (2014). Grey Wolf Optimizer. *Advances in Engineering Software*, *69*, 46-61.

Molaei, S., Moazen, H., Najjar-Ghabel, S., and Farzinvash, L. (2021). Particle swarm optimization with an enhanced learning strategy and crossover operator. *Knowledge-Based Systems*, *215*, 106768.

Nagra, A. A., Han, F., Ling, Q., and Mehta, S. (2019). An Improved Hybrid Method Combining Gravitational Search Algorithm With Dynamic Multi Swarm Particle Swarm Optimization. *IEEE Access*, *7*, 50388-50399.

Nickabadi, A., Ebadzadeh, M. M., and Safabakhsh, R. (2011). A novel particle swarm optimization algorithm with adaptive inertia weight. *Applied soft computing*, *11*(4), 3658-3670.

Pitchaimanickam, B., and Murugaboopathi, G. (2020). A hybrid firefly algorithm with particle swarm optimization for energy efficient optimal cluster head selection in wireless sensor networks. *Neural Computing and Applications*, *32*(12), 7709-7723.

Rauf, H. T., Bangyal, W. H. K., and Lali, M. I. (2021). An adaptive hybrid differential evolution algorithm for continuous optimization and classification problems. *Neural Computing and Applications*, *33*(17), 10841-10867.

Rauf, H. T., Shoaib, U., Lali, M. I., Alhaisoni, M., Irfan, M. N., and Khan, M. A. (2020). Particle Swarm Optimization With Probability Sequence for Global Optimization. *IEEE Access*, *8*, 110535-110549.

Raza, A., Yousaf, Z., Jamil, M., Gilani, S. O., Abbas, G., Uzair, M., Shaheen, S., Benrabah, A., and Li, F. (2018). Multi-Objective Optimization of VSC Stations in Multi-Terminal VSC-HVdc Grids, Based on PSO. *IEEE Access*, *6*, 62995-63004.

Rehman, A. U., Islam, A., and Belhaouari, S. B. (2020). Multi-cluster jumping particle swarm optimization for fast convergence. *IEEE Access*, *8*, 189382-189394.

Sallam, K. M., Elsayed, S. M., Chakrabortty, R. K., and Ryan, M. J. (2020). Improved multi-operator differential evolution algorithm for solving unconstrained problems. *2020 IEEE Congress on Evolutionary Computation (CEC)*, 1-8.

Sandgren, E. (1990). Nonlinear Integer and Discrete Programming in Mechanical Design Optimization. *Journal of Mechanical Design*, *112*(2), 223-229.

Schindl, D., and Zufferey, N. (2015). A learning tabu search for a truck allocation problem with linear and nonlinear cost components. *Naval Research Logistics (NRL)*, *62*(1), 32-45.

Sedighizadeh, D., and Mazaheripour, H. (2018). Optimization of multi objective vehicle routing problem using a new hybrid algorithm based on particle swarm optimization and artificial bee colony algorithm considering Precedence constraints. *Alexandria engineering journal*, *57*(4), 2225-2239.

Shi, Y., and Eberhart, R. (1998). A modified particle swarm optimizer. *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*, 69-73.

C. T. Yue, K. V. Price, P. N. Suganthan, J. J. Liang, M. Z. Ali, B. Y. Qu, N. H. Awad and P. Biswas, Problem Definitions and Evaluation Criteria for the CEC 2020 Special Session and Competition on Single Objective Bound Constrained Numerical Optimization, Technical Report 201911, Computational

Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore, November 2019.

Taherkhani, M., and Safabakhsh, R. (2016). A novel stability-based adaptive inertia weight for particle swarm optimization. *Applied Soft Computing*, *38*, 281-295.

Tang, H., Chen, R., Li, Y., Peng, Z., Guo, S., and Du, Y. (2019). Flexible job-shop scheduling with tolerated time interval and limited starting time interval based on hybrid discrete PSO-SA : An application from a casting workshop. *Applied Soft Computing*, *78*, 176-194.

Ullah, Z., Wang, S., Radosavljević, J., and Lai, J. (2019). A Solution to the Optimal Power Flow Problem Considering WT and PV Generation. *IEEE Access*, *7*, 46763-46772.

Wang, Y., Zhang, H., and Zhang, G. (2019). cPSO-CNN : An efficient PSO-based algorithm for fine-tuning hyper-parameters of convolutional neural networks. *Swarm and Evolutionary Computation*, *49*, 114-123.

Xia, X., Gui, L., He, G., Wei, B., Zhang, Y., Yu, F., Wu, H., and Zhan, Z.-H. (2020). An expanded particle swarm optimization based on multi-exemplar and forgetting ability. *Information Sciences*, *508*, 105-120.

Xia, X., Gui, L., He, G., Xie, C., Wei, B., Xing, Y., Wu, R., and Tang, Y. (2018). A hybrid optimizer based on firefly algorithm and particle swarm optimization algorithm. *Journal of Computational Science*, *26*, 488-500.

Xia, X., Gui, L., Yu, F., Wu, H., Wei, B., Zhang, Y.-L., and Zhan, Z.-H. (2019). Triple archives particle swarm optimization. *IEEE transactions on cybernetics*, *50*(12), 4862-4875.

Xia, X., Gui, L., and Zhan, Z.-H. (2018). A multi-swarm particle swarm optimization algorithm based on dynamical topology and purposeful detecting. *Applied Soft Computing*, *67*, 126-140.

Xia, X., Xing, Y., Wei, B., Zhang, Y., Li, X., Deng, X., and Gui, L. (2019). A fitness-based multi-role particle swarm optimization. *Swarm and evolutionary computation*, *44*, 349-364.

Xu, G., Cui, Q., Shi, X., Ge, H., Zhan, Z.-H., Lee, H. P., Liang, Y., Tai, R., and Wu, C. (2019). Particle swarm optimization based on dimensional learning strategy. *Swarm and Evolutionary Computation*, *45*, 33-51.

Yalcinoz, T., and Rudion, K. (2019). Economic Load Dispatch Using an Improved Particle Swarm Optimization based on functional constriction factor and functional inertia weight. *2019 IEEE International Conference on Environment and Electrical Engineering and 2019 IEEE Industrial and Commercial Power Systems Europe (EEEIC / IandCPS Europe)*, 1-5.

Yang, J., Wang, X., and Bauer, P. (2019). Extended PSO Based Collaborative Searching for Robotic Swarms With Practical Constraints. *IEEE Access*, *7*, 76328-76341.

Yang, X.-S. (2010). *Engineering optimization : An introduction with metaheuristic applications*. John Wiley and Sons.

Zhang, Y., Liu, X., Bao, F., Chi, J., Zhang, C., and Liu, P. (2020a). Particle swarm optimization with adaptive learning strategy. *Knowledge-Based Systems*, *196*, 105789.

Zhang, Y., Liu, X., Bao, F., Chi, J., Zhang, C., and Liu, P. (2020b). Particle swarm optimization with adaptive learning strategy. *Knowledge-Based Systems*, *196*, 105789.

Chapter 2

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

2.1 Introduction

The global economy and prosperity depend critically on manufacturing (Li and Meerkov, 2009). In the last decades, many papers have been devoted to serial production lines, which are widely employed in manufacturing systems. These lines consist of machines often grouped in sequential order with buffers between each pair of adjacent machines. Units/items (material, parts, or products) flow through these machines according to predefined sequences. Most of the manufacturing literature aims to investigate ways to increase the effectiveness of these production lines considering various objectives (e.g., production rate, profit). However, the material flow may be disrupted by stochastic factors (e.g., machines failure and repair, random service time, starving and blocking events), reducing thus the efficiency of the production line. Allocating additional buffer sizes raises the average production rate (PR) of the line and reduces the disruption propagation. However, it also increases the work-in-process (WIP) inventories, the capital investment cost, and the floor space. Thus, determining the appropriate buffer allocation to reduce manufacturing costs while maintaining a desirable PR is a challenging manufacturing optimization problem.

In a serial production line, this challenge is called the buffer allocation problem (BAP). It is one of the most important topics for the production line designers. Recently, Demir et al. (2014) and Weiss et al. (2019) have reviewed the related literature. It has been reported, according to the topology of the line and the solution methodology, that there are various versions of BAPs, including the primal/dual problem, the constraint/unconstraint profit maximization, and the minimization of the WIP. The production line can be composed of reliable machines (i.e., the machines are not subject to failure) or unreliable machines (i.e., the machines are subject to failure). The solution method involves the development of algorithms based on evaluative search (e.g., simulation, decomposition method, Markovian state model) or generative search (e.g., dynamic programming, search methods, metaheuristics such

genetic algorithm, simulated annealing, and particle swarm optimization). Concerning the considered objective function and constraints, the primal problem minimizes the total buffer capacity required for the line with a production rate constraint, whereas the dual problem maximizes the production rate achievable subject to a given total buffer capacity. For the constraint profit maximization, the aim is to maximize the profit with a given total buffer capacity, and the latter is relaxed for the unconstraint profit maximization.

The BAP is a well-known NP-hard combinatorial optimization problem (Dolgui et al., 2013; Smith and Cruz, 2005), and there is no closed-form formula among decision variables (e.g., the buffer capacities) and performance measures when the production line consists of more than two machines (Xi et al. 2020). Furthermore, as the size of the problem increases, the computational complexity of the problem increases considerably. Solving the BAP efficiently and reliably remains a challenging problem for large production lines.

For solving the BAP, the main contributions of this chapter are to investigate the performance of particle swarm optimization (PSO) and to develop a novel variant of PSO, named Adaptive Dynamic Jumping Particle Swarm Optimization (APSO for short). The objective function consists in maximizing the production rate of unreliable serial production lines. APSO integrates a leaping strategy based on both logarithmic and exponential functions and dynamic parameters. To the best of our knowledge, only two studies have proposed PSO approaches for the BAP (Narasimhamu et al., 2014, 2015). Another goal of this work is to develop (near-)optimal solutions for different system sizes and various configurations that may facilitate designers in making decisions related to the production line design.

The remainder of this chapter is organized as follows. Section 2 presents the literature review of the solution approaches for the BAP. Section 3 formulates the problem mathematically. Section 4 describes the optimization method and the related assumptions. Various numerical experiments are discussed in Section 5. Finally, the chapter ends up with a conclusion and future research directions.

2.2 Literature review

Both generative and evaluative tools are needed to solve the BAP (Spinellis and Papadopoulos, 2000). The former generates solutions, whereas the latter evaluates their performance.

Concerning evaluation methods, various approaches have been developed to measure the performance of the production line. They can be classified in two categories: analytical methods and simulation.

- Markovian state models (Diamantidis and Papadopoulos, 2009) are an example of exact analytical techniques that are used to comprehensively characterize the features of the system, but they are only suitable for small production lines due to their enormous state space and intractable computing time.
- Simulations methods (Alfieri and Matta, 2012; Amiri and Mohtashami, 2012) are flexible tools that use a simulation package (e.g., Arena) for the performance evaluation of large and complex realistic production systems. The simulation approach is typically employed to solve, at a detailed level, design challenges, whereas other assessment techniques might not be as precise in practical applications (Demir et al., 2014). However, simulations are time consuming (Helber et al., 2011).

For larger production lines, three well-known approximation methods are widely employed: the generalized expansion method (Spinellis et al., 2000; Zhang et al., 2017), the decomposition method (Diamantidis et al., 2020; Liberopoulos, 2018; Massim et al., 2010), and the aggregation method (Dolgui et al., 2002; Zhao et al., 2016).

- The generalized expansion methods are based on queuing models, and they can be applied for serial production lines as well as for split and merge configurations. They can be used for reliable machines and distributed service times.
- To reduce the computational effort, the decomposition methods divide the entire system into smaller subsystems that are easier to tackle. These methods, unlike the generalized expansion methods, are used under the assumptions that service times (resp. repair/failure rates) are either exponentially distributed or deterministic (resp. exponentially distributed or geometric).
- Finally, the idea of the aggregation methods is to replace each two-machine-and-one-buffer component with a single equivalent machine. This aggregation is repeated in forward and backward fashions through recursive procedures until the last or the first station is reached.

Regarding the generative (optimization) methods, an extensive amount of research has been devoted to finding the (near-)optimal values of the decision variables (Papadopoulos et al., 2013). The simplest generative method is Complete Enumeration (CE), but it is suitable only for small systems. Although this problem has been addressed in various works, no approach can solve it for large production lines in a computationally efficient manner (Hillier and SO, 1991; Zhou and Lian, 2011). To improve efficiency and deal effectively with the combinatorial nature of the BAP, various optimization methods have been proposed, including dynamic programming, search methods and metaheuristics.

- Dynamic programming is a well-known powerful optimization method (Diamantidis and Papadopoulos, 2004; Huang et al., 2002; Yamashita and Altiok, 1998) that divides the BAP into many subproblems to reduce the problem combinatorial complexity. However, some restrictive assumptions are usually made when solving the BAP (e.g., it is applied for reliable production lines only).
- Search methods tend to choose feasible solutions that are close to (near-)optimal solutions to address the combinatorial explosion of feasible solutions (Demir et al., 2014). Commonly used search methods include the gradient method (Shi and Gershwin, 2016; Gershwin and Schor, 2000), which relies on the monotonicity between productivity and buffer size using a gradient variable to ensure the choice of the best option. Other search methods involve the degraded ceiling method (Nahas et al., 2006), and the Hooke and Jeeves method (Papadopoulos and Vidalis, 1998). Such search methods have two main disadvantages: they are easily trapped in local optima; it is difficult to determine the impact of small changes in buffer capacities on the global performance of the system. Therefore, various researchers have developed heuristic search algorithms (Sabuncuoglu et al., 2006).
- Metaheuristics use specific techniques for managing the search process and exploring the solution space while maintaining reasonable computing times. Typical metaheuristics include genetic algorithm (GA) (Cruz et al., 2018; Dolgui et al., 2002), simulated annealing (SA) (Spinellis and Papadopoulos, 2000), tabu search (TS) (Costa et al., 2015; Demir et al., 2011), ant colony optimization (Yegul et al., 2017). To improve the efficiency of the traditional metaheuristics, a novel trend is to hybridize them with other methods. The following combinations can be given: GA and SA (Kose and Kilincci, 2015); GA and Finite Perturbation analysis (FPA) (Kassoul et al., 2022a), GA and Branch and Bound (Dolgui et al., 2007); TS and Nested Partitions (Shi and

Men, 2003); PSO and Optimal Computing Budget Allocation (OCBA) (Lin and Chiu, 2018).

Regarding PSO, only two papers can be found for the BAP (Narasimhamu et al. (2014, 2015)), considering instances with up to 8 machines. In addition, the study of Lin and Chiu (2018) hybridize PSO with OCBA, but for the Resource Allocation Problem. With respect to the literature, we will show that the proposed APSO can efficiently address the BAP in unreliable serial production lines for instances ranging from 3 to 100 machines. In addition, we will discuss its efficient convergence thanks to the integration of a leaping strategy in the velocity equation (using two components based on exponential and logarithmic functions). This allows particles to make leaps and thus escape from a possible local optimum. Determining the role and action set of each particle is a very sensitive task, as it was the case for other optimization paradigms involving "agents" (Zufferey, 2012).

2.3 Problem formulation

Serial production lines (also denoted as transfer lines, tandem lines, flow lines, etc.) are often composed of several machines in which material (parts) flow from outside the system to the first machine M_1, then move to the first buffer b_1, then to machine M_2 and so on until they reach the last machine M_n and leave the system. An example of such a production line is presented in Figure 2.1, where n machines $(M_1, M_2, \ldots, M_n)$ are connected between them and separated by $(n\text{-}1)$ buffers $(b_1, b_2, \ldots, b_{n-1})$; and where b_i is the buffer capacity situated between two adjacent machines M_i and M_{i+1}. For each machine, we also know its repair (r_i) and failure (f_i) probabilities, as well as the service time (t_i) to process one unit.

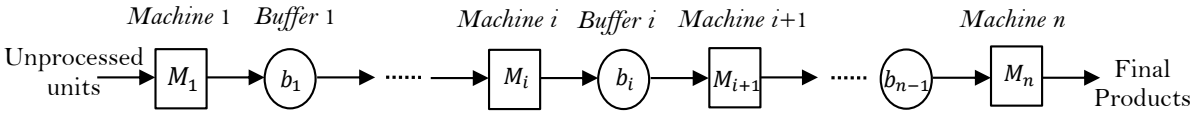


Figure 2.1. Serial production line

As machines are unreliable, at time t, the state of a machine can be either UP (it may execute operations on parts) or DOWN (i.e., it cannot execute any operation due to internal failure). We work under the following assumptions: the first machine is never starved, and the last machine is never blocked; the machines are unreliable and can fail only when it is UP and neither blocked nor starved; the repair and failure rates of machines follow a geometric distribution; delays in supplying parts when moving from stocks to machines and conversely

are negligible; when a machine is UP, it can be starved or Null Input (NI) (resp. blocked or Full Output (FO)) if its upstream (resp. downstream) buffer is empty (resp. full).

We define the repair (r_i) and the failure (f_i) probabilities of machine i as follows: r_i = probability that machine i is UP for the next part, knowing that it is DOWN for the current part; f_i = probability that machine i is DOWN for the next part, knowing that it is UP for the current part. The Mean Time Between Failures of machine i with service time t_i is defined as $MTBF_i = t_i/f_i$, whereas its Mean Time To Repair is defined as $MTTR_i = t_i/r_i$.

In line with Figure 2.1, each b_i is the buffer size right after machine M_i, and it must be positive and integer. Let B_{max} be the total buffer capacity that is available to be dispatched over the buffers $b_1, b_2, \ldots, b_{n-1}$ of the serial production line. The considered BAP optimization problem consists in determining the vector $b = (b_1, b_2, \ldots, b_{n-1})$ to maximize the average PR while respecting $\sum_{i=1}^{n-1} b_i = B_{max}$.

2.4 Solution method

2.4.1 Particle Swarm Optimization (PSO)

PSO is a metaheuristic proposed by Kennedy and Eberhart (1995). It is inspired from the social collective behaviour of birds/fishes flocking, where each particle (potential solution) learns from its own experience (*personal* best position) and from the experience of other particles (*global* best position) during the repeated moves in the search space to find a global optimum. Each particle is defined by two parameters: velocity and position. In the standard PSO, at every iteration (generation) t, the position x_i^t and the velocity v_i^t of each particle i are updated as in Equations (2.1) and (2.2), where w is the inertia weight coefficient, c_1 (the *cognitive* parameter) and c_2 (the *social* parameter) are acceleration coefficients, and R_1 and R_2 are two random values taken in the interval [0,1], and it will be also the case below for $R_3, R_4, \ldots, R_{10}$.

$$v_i^{t+1} = w\, v_i^t + c_1 R_1 \left(p_{best_i}{}^t - x_i^t\right) + c_2\, R_2 \left(p_{best_i}{}^t - x_i^t\right) \quad (2.1)$$

$$x_i^{t+1} = v_i^{t+1} + x_i^t \quad (2.2)$$

2.4.2 Parameters and generation of the initial swarm of APSO

The parameters of the proposed APSO method are the following.

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

The initial swarm is composed of m different particles (or solutions or configurations) P = $(P_1, P_2, \ldots, P_m)$, where each particle $P_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,j}, \ldots, p_{i,n-1})$ represents $(n-1)$ buffer capacities, and $p_{i,j}$ denotes the j^{th} buffer (position) of the i^{th} particle. Each solution is generated randomly and uniformly to cover various regions of the search space, using the discrete uniform distribution. To satisfy the B_{max} constraint, an adjustment procedure could reduce/increase uniformly some buffer values of the particle. The reader is referred to (Kassoul et al., 2022a) for more details on the generation of such initial solutions.

2.4.3 Proposed APSO

The proposed method is based on the combination of two recent studies, namely DCS-PSO (Kassoul et al., 2021) and ExPSO (Kassoul et al., 2022b). The first purpose is to develop a jumping strategy to overcome the limitations of classical PSO (e.g., premature convergence). This strategy includes two components (i.e., a logarithm part and an exponential part). The second purpose is to control the velocity for a faster convergence using dynamic parameters. The adaptation of DCS-PSO and ExPSO to our problem is presented in the two following subsections, and their combination is proposed afterwards.

2.4.3.1 Logarithmic component: DCS-PSO

DCS-PSO is presented in Figure 2.2. It is characterized by its avoidance of the usage of the inertia weight w and by emphasizing the cognitive and social aspects of PSO while still regulating the velocity values. Thus, dynamic functions and parameters are proposed to avoid falling in local minima (see Kassoul et al., (2021) for more details). A new cognitive function ζ^t is introduced by integrating a logarithmic function into the equation of the velocity as follows, where ϵ is a very small positive value.

$$\zeta^t = log\left(\left\|{p_{best}}_i^t - x_i^t\right\| + \epsilon\right) \tag{2.3}$$

2.4.3.2 Exponential component: ExPSO

ExPSO is presented in Figure 2.3, where R_{vec} is a random vector where each component has its value in [0,1]. It is characterized by the decomposition of the swarm population into three equal subswarms (see Kassoul et al. (2022b) for more details). The diversity of the population is maintained, and new areas of the solution space are explored through the merging of personal and global worst positions into the equation of the velocity. To overcome a premature convergence, this variant of PSO integrates a new component based on an exponential function defined as follows, where ϵ is a very small positive value.

$$\xi^t = e^{1/(\|p_i^t - x_i^t\| + \epsilon)} \tag{2.4}$$

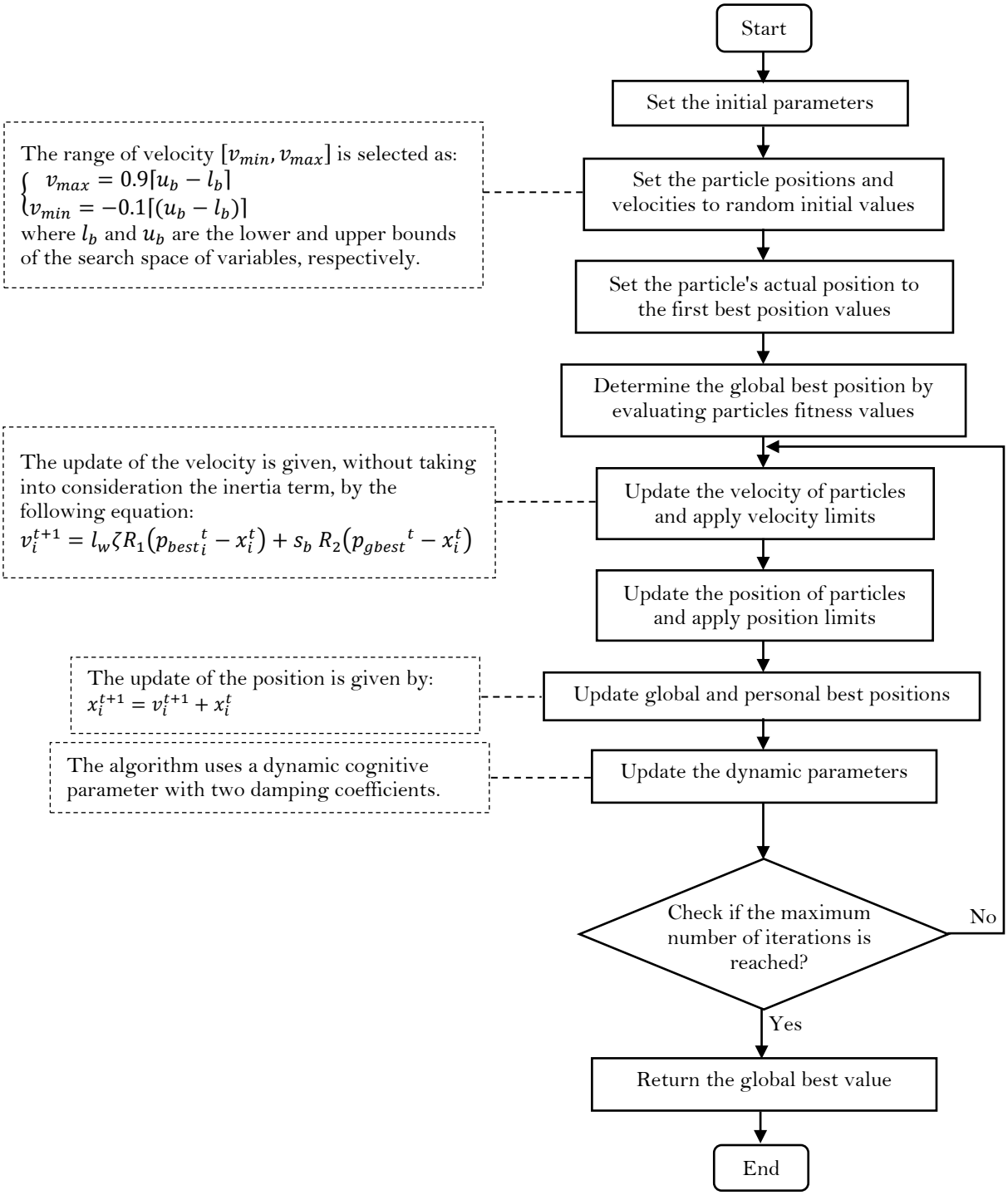


Figure 2.2. Flowchart of DCS-PSO

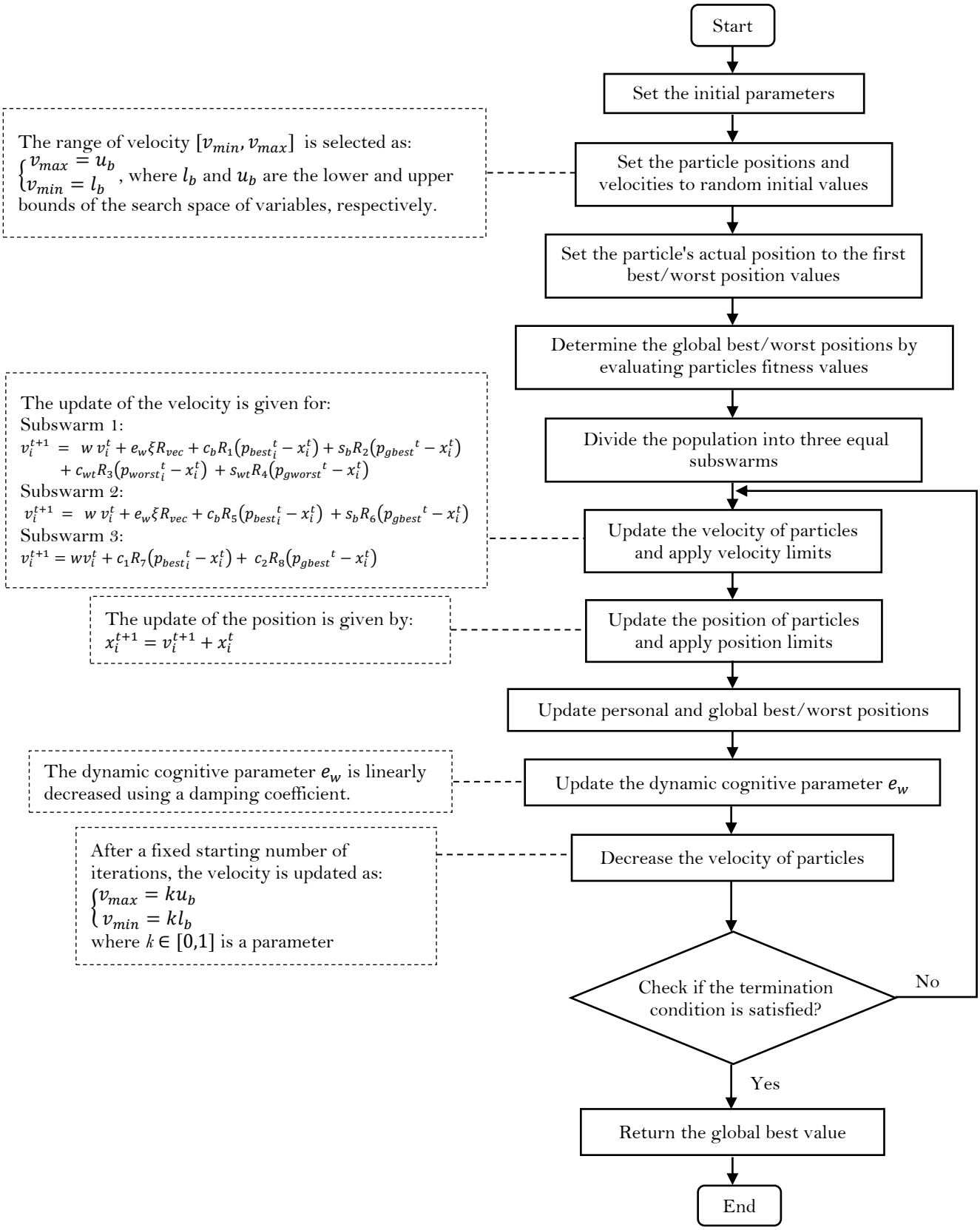


Figure 2.3. Flowchart of ExPSO

2.4.3.3 Expo-logarithm algorithm: APSO

The pseudocode of APSO is presented in Algorithm 2.1. A novel learning strategy is thus proposed by embedding the presented logarithmic and exponential components into the updating equation presented in Equation (2.5). In other words, APSO combines the above-presented DCS-PSO and ExPSO.

The first step of APSO is the generation of the initial population of swarm (see Subsection 2.4.2) and the random generation of the velocities of particles using the discrete uniform distribution $U\{v_{min}, v_{max}\}$ for each coordinate, where v_{min} and v_{max} are set to the lower and upper bounds of the search space of variables (i.e., l_b and u_b), respectively.

The second step of the algorithm is the decomposition of the swarm population into three equal subswarms N_1, N_2 and N_3. The velocity of the first subswarm N_1 is formulated in Equation (2.5), where ξ and ζ are the exponential and logarithmic components defined above.

$$v_i^{t+1} = w\, v_i^t + e_w \xi R_{vec} + l_w \zeta R_1\left(p_{best_i}{}^t - x_i^t\right) + c_b R_2\left(p_{best_i}{}^t - x_i^t\right) + s_b R_3\left(p_{gbest}{}^t - x_i^t\right) + c_{wt} R_4\left(p_{worst_i}{}^t - x_i^t\right) + s_{wt} R_5\left(p_{gworst}{}^t - x_i^t\right) \quad (2.5)$$

The benefit of the use of the logarithmic and exponential components is to allow particles to make jumps when the velocity is near zero (i.e., when the particle is possibly trapped in a local optimum) and thus escaping from these regions. Furthermore, the exploration and exploitation phases must be well balanced to perform global and local searches efficiently. For this aim, the exponential weight parameter e_w is used to identify new regions in the search space. This parameter decreases linearly using the damping coefficient d, permitting particles to explore the whole search space with large leaps at the beginning of the search and to improve the solution with small leaps in particular regions of the solution space. The exponential weight parameter e_w is given as:

$$e_w = d\, e_w \quad (2.6)$$

The update of the velocity of the first subswarm contains the global and personal worst positions to maintain a diversity of the population and to investigate new solution-space regions. This could help to avoid possible premature convergence. We note that to investigate opposite regions visited by the personal and global worst positions, the coefficient values of these two positions (personal/global worst) have negative values (i.e., the cognitive acceleration worst-coefficient c_{wt} and the social acceleration worst-coefficient s_{wt}).

For the second subpopulation N_2, the velocity is influenced by logarithmic and exponential components, the previous velocity, and the global and personal best positions. Thus, the velocity is formulated as:

$$v_i^{t+1} = w\, v_i^t + e_w \xi R_{vec} + l_w \zeta R_6\left(p_{best_i}{}^t - x_i^t\right) + c_b R_7\left(p_{best_i}{}^t - x_i^t\right) + s_b\, R_8\left(p_{gbest}{}^t - x_i^t\right) \quad (2.7)$$

Algorithm 2.1. APSO

Step 1: Initialization

- Set the initial parameters $e_w, l_w, c_b, s_b, c_{wt}, s_{wt}, c_1, c_2, d, w, N, MaxG$
- Initialize the position of the particles x_i $(i = 1,2,\dots,N)$
- Initialize uniformly and randomly the velocity v_i $(i = 1,2,\dots,N)$.
- Determine the fitness value of particles (PR) using the discrete-event simulation model and set the first best and worst position values, p_{best_i} and p_{worst_i}, equal to the actual positions of particles x_i $(i = 1,2,\dots,N)$
- Determine the global best and worst positions p_{gbest} and p_{gworst}

Step 2: Update particle position and velocity

- Divide the population into three equal subswarms N_1, N_1 and N_3

  for the first subswarm N_1, **do**

  - $\xi = e^{1/(\|p_{best_i} - x_i\| + \epsilon)}$, $\zeta = log\left(\left\|p_{best_i} - x_i\right\| + \epsilon\right)$
  - Compute each v_i as in Equation (2.5)

  for the second subswarm N_2, **do**

  - Compute each v_i as in Equation (2.7)

  for the third subswarm N_3, **do**

  - Compute each v_i as in Equation (2.8)
- Apply velocity limits using the equations $v_{i,j} = \max\left(v_{i,j}; v_{min}\right)$ and $v_i = \min\left(v_{i,j}; v_{max}\right)$, where $v_{i,j}$ denotes the j^{th} velocity coordinate of the i^{th} particle
- Update the position of particles using the equation $x_i = x_i + v_i$
- Apply position limits using the equations $x_{i,j} = \max\left(x_{i,j}; l_b\right)$ and $x_i = \min\left(x_{i,j}; u_b\right)$, where $x_{i,j}$ denotes the j^{th} position coordinate of the i^{th} particle

Step 3: Update global and personal best/worst positions

- Calculate particles' fitness values (PR)
- Update the global/personal best/worst positions of the particles
- Update the dynamic parameter *a* using Equation (2.6)
- Update the inertia weight w using Equation (2.9)

Step 4: Procedure of convergence

- Check if the termination criterion is satisfied (i.e., the maximum number of iterations is reached)

Finally, the velocity of the subswarm N_3 is updated using the C_NPSO procedure proposed by Kassoul et al. (2022b). It differs from the standard PSO by taking $c_1 = -1$ and $c_2 = 2$. The equation of velocity is formulated as:

$$v_i^{t+1} = wv_i^t + c_1 R_9\left(p_{best_i}{}^t - x_i^t\right) + c_2 R_{10}\left(p_{gbest}{}^t - x_i^t\right) \quad (2.8)$$

The value of w used in APSO decreases linearly from an initial value to a final value. This decreases continuously the ability of particles to global search and at the same time, increases continuously their ability to local search. The proposed w is updated as:

$$w = d\left(\frac{1-w}{1+w}\right) \quad (2.9)$$

2.5 Numerical Experiments

All experiments are carried out using a computer with 16 GB of RAM and a processor of 2.4-GHz Core (TM) i5 CPU. The algorithms are implemented in Java and the discrete-event simulation models are developed using the Arena simulation language V14.0 to calculate the average PR. APSO stops when reaching $MaxG = 50$ generations.

We consider three instance sizes for serial production lines: small lines (n in {3, 5, 10} machines), medium lines ($n = 20$) and large lines (n in {40, 100}). All machines are unreliable and identical. The repair and failure times are geometrically distributed with $MTTR = 1/r$ and $MTBF = 1/f$, where r and f are the repair and failure probabilities, respectively. All machines have a service time $t_i = 1$. The buffer allocation, the average PR values and the convergence tendency will be discussed.

The BAP in serial production lines has been widely studied in the literature, and we compare APSO with seven state-of-the-art algorithms for which the data and the results are available for some of the considered instances. If the results of a method are not given for an instance, it simply means that such results are not available in the literature.

- GT is a Gradient Technique (Ho et al., 1979),
- DGT is a Dual Gradient Technique (Gershwin and Schor, 2000),
- DC relies on a Degraded Ceiling method and a decomposition approximation (Nahas et al., 2006),

- IDA is an Immune Decomposition Algorithm (Massim et al., 2010),
- ADA-TS is a Tabu Search with an Analytical Decomposition Approximation (Demir et al., 2011),
- GA-SA combines Simulated Annealing and a Genetic Algorithm (Kose and Kilincci, 2015),
- GA-FPA combines a Genetic Algorithm and Finite Perturbation Analysis (Kassoul et al., 2022a) (This method will be presented in detail in Chapter 3).

It is important to note here that GA-FPA was showed to be a very efficient algorithm for the BAP involving serial production lines. The detailed comparisons with various state-of-the-art methods are presented in (Kassoul et al., 2022a).

Table 2.1 presents the parameters of APSO. In the next tables, the results of the best algorithm are marked in bold. Figure 2.4 provides the representative PR convergence curves of APSO for some instances with n in {5, 10, 20, 40}.

Table 2.1. Parameter settings of APSO

Parameter Settings			
Logarithmic weight parameter	$l_w = 2$	Social scaling parameter	$c_2 = 2$
Exponential weight parameter	$e_w = 2$	Inertia weight	$w = 0.9$
Cognitive acceleration best-coefficient	$c_b = 2$	Damping coefficient	$r = 0.9$
Social acceleration best-coefficient	$s_b = 2$	Decreasing velocity coefficient	$k = 0.2$
Cognitive acceleration worst-coefficient	$c_{wt} = -1$	Number of particles of subpopulation 1	$N_1 = N/3 = 10$
Social acceleration worst-coefficient	$s_{wt} = -1$	Number of particles of subpopulation 2	$N_2 = N/3 = 10$
Cognitive scaling parameter	$c_1 = -1$	Number of particles of subpopulation 3	$N_3 = N/3 = 10$

Table 2.2 presents the number of simulated units/parts and runs (replications) used in APSO. Such numbers have been chosen for a fair comparison (i.e., comparable values are used in the other methods) and to have stable results (i.e., the average PR values are consistently obtained with such settings).

Table 2.2. Number of simulated units and number of replications used in APSO

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

The computation times are not discussed in detail (except for the very large instance with n = 100 machines), but on average, considering a large instance of n = 40 machines (which covers many industrial cases), the order of magnitude to get the best solution is 96 minutes, which is reasonable according to various industrial studies in production (e.g., Respen et al., 2016, 2017; Zufferey, 2016).

2.5.1 Results on a small instance with 3 machines

Table 2.3 gives the repair and failure rates for this instance proposed in (Gershwin and Schor, 2000), where the buffer configuration and PR are however not mentioned explicitly. Table 2.4 compares APSO, GA-FPA and IDA. We can observe that APSO and GA-FPA find the same PR. The three methods obtain a similar allocation scheme, where the first buffer is larger than the second. This could be explained by the large values of f_1 and r_1 when compared to f_2 and r_2.

Table 2.3. 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 2.4. Results on an instance with n = 3 machines and B_{max}= 20

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

2.5.2 Results on a small instance with 5 machines

Table 2.5 gives the repair and failure rates for this instance proposed in (Ho et al., 1979). Table 2.6 presents the results for APSO, GA-FPA, GA-SA, ADA-TS, IDA, DGT, and GT. APSO obtains the best PR value. This best configuration is obtained before 15 generations (see Figure 2.4). All the configurations found by the different approaches allow smaller buffer slots at the ends of the line. This could lead to a better passage of the parts and thus avoid an eventual blockage of the line.

Table 2.5. 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 2.6. Results on an instance with $n = 5$ machines and $B_{max} = 31$

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

2.5.3 Results on a small instance with 10 machines

Table 2.7 gives the repair and failure rates for this instance proposed by Nahas et al. (2006). Table 2.8 presents the results for APSO, GA-FPA, GA-SA, ADA-TS, IDA, and DC. From the results of Table 2.8, it can be observed that APSO obtains the second-best PR value, with its best buffer allocation reached before 25 generations (see Figure 2.4).

Table 2.7. Parameters of the considered instance

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

Table 2.8. Results on an instance with $n = 10$ machines and $B_{max} = 270$

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

2.5.4 Results on medium instances with 20 machines

Columns 1 and 2 of Table 2.9 gives the repair and failure rates for this instance proposed by Demir et al. (2011). All machines have similar repair and failure rates ranging from 0.1 to 0.9 with an increment of 0.1 (it means that 9 cases are considered here). The other columns give

the PR values obtained by the following methods: APSO, GA-FPA, GA-SA, and ADA-TS. We can observe that APSO outperforms GA-SA and ADA-TS. For this instance, APSO and GA-FPA obtain a similar performance (they both obtain the best results on four cases). For APSO, the best PR values are obtained before generation 30. As expected, for all methods, PR augments with r_i and f_i.

Table 2.9. Results on an instance with $n = 20$ machines and $B_{max} = 100$

Parameters		**PR value for the considered methods**			
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

2.5.5 Results on large instances with 40 machines

Table 2.10 has the same structure as Table 2.9 for an instance proposed in (Demir et al., 2011). The same methods are compared. For this much larger instance, APSO is substantially more efficient than the other methods. Indeed, it provides the best solution in 7 out of 8 cases. This seems to indicate that the efficiency of APSO augments with the instance size, when compared to other methods. As for the previous instance, PR augments with r_i and f_i.

Table 10. Results on an instance with $n = 40$ machines and $B_{max} = 200$

Parameters		**PR value for the considered methods**			
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

2.5.6 Results on a large instance with 100 machines

Only few studies have been devoted on BAPs for very large production lines (e.g., Spinellis et al., 2000; Kose and Kilincci, 2015). According to researchers in this field, the computation times required to get competitive results are extremely long (typically dozens of hours). In other words, the computation time grows considerably with the number n of machines. We consider here a production line with $n = 100$ unreliable machines proposed by Kose and Kilincci (2015). The repair and failure rates are geometrically distributed with $MTTR = 2$ and $MTBF = 20$, respectively. Given the very high required computing times, the experiment is conducted for $MaxG = 20$ generations (instead of 50). Table 2.11 presents the comparative results of APSO and GA-SA. The computing times are provided in seconds in the last column. We can observe that APSO significantly outperforms GA-SA according to both quality (i.e., PR values) and speed (i.e., computing time requested to obtain its best solution). This is likely to indicate the good trade-off of APSO among solution quality and computing time. This feature may help a decision-maker to choose an acceptable search time to design a production line.

Table 2.11. Results on an instance with $n = 100$ machines and $B_{max} = 300$

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

2.5.7 Discussion and managerial insights

Regarding the augmentation of *n*, the following observations can be made when observing Figure 2.4 and Tables 2.4, 2.6, 2.8, 2.9 and 2.10. APSO becomes more competitive when compared to the other methods, and it usually obtains its best solutions later while keeping some gap with the allowed number of generations $MaxG = 50$ (in Figure 2.4: at generation 11 with $n = 5$, at generation 20 with $n = 10$, at generation 25 with $n = 20$, at generation 22 with $n = 40$). This is likely to indicate a high efficiency for different problem sizes, which could be explained by an efficient balance among exploitation and exploration search abilities during the search process.

Regarding the allocation of buffer capacity for such different production line sizes, APSO generates the best allocation of buffers in 14 out of the total 22 cases, and very competitive results when it is not ranked first. However, the patterns are often unexpected, which highlights how challenging the issue we tackled is. It has been noted in this study that the middle buffers values are almost similar and are relatively bigger than the buffer capacities at

the ends of the line. Moreover, as the number of machines increases, the buffers values tend to be the same. Although depending on the context, these allocation patterns are among the most crucial insights for decision makers when designing production lines.

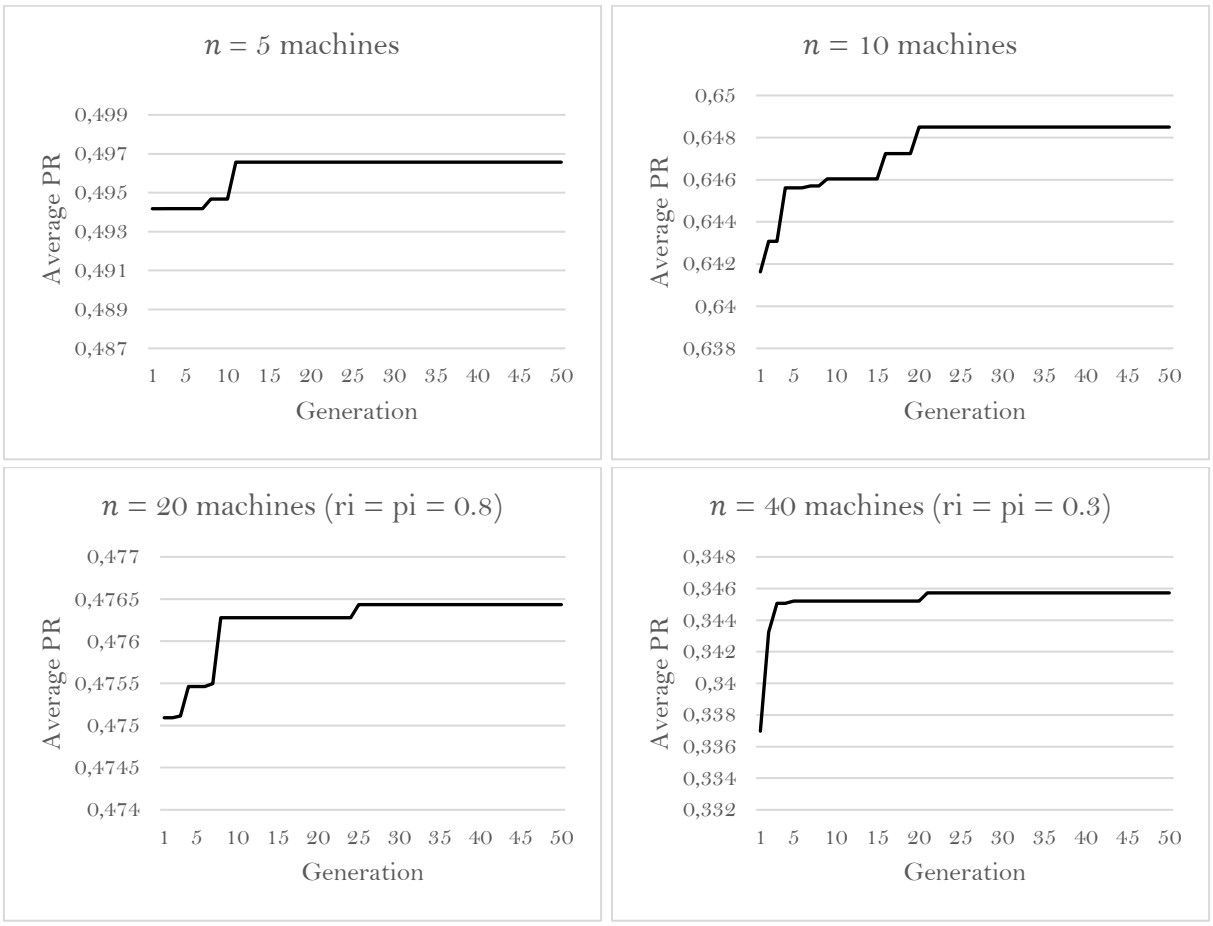


Figure 2.4. Representative convergence curves of APSO for some instances with n in {5, 10, 20, 40}

2.6 Conclusion

In this study, a fast, efficient, and reliable algorithm (called APSO) is proposed for the buffer allocation problem in unreliable serial production lines to maximize the production rate. APSO is a variant of particle swarm optimization, using a jumping strategy with logarithmic and exponential functions, along with dynamic parameters that are integrated into the equation of the velocity. This strategy allows the algorithm to efficiently explore the solution space and to converge quickly to competitive solutions. The experiments conducted for various instance sizes (ranging from $n = 3$ to $n = 100$ machines) show that APSO does not

suffer from the increase in n. Indeed, it outperforms state-of-the-art methods in terms of solution quality and computing time. As future research directions, we may consider a hybridization of APSO with other metaheuristics, including the integration some refined learning mechanism as proposed by Schindl and Zufferey (2015) and by Thevenin and Zufferey (2019) in the context of local-search approaches. It would also be advantageous to expand the proposed method to more complex production lines, including split systems and assembly systems.

2.7 References

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

Amiri, M., and Mohtashami, A. (2012). Buffer allocation in unreliable production lines based on design of experiments, simulation, and genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, *62*(1), 371-383.

Costa, A., Alfieri, A., Matta, A., and Fichera, S. (2015). A parallel tabu search for solving the primal buffer allocation problem in serial production systems. *Computers and Operations Research*, *64*, 97-112.

Cruz, F. R., Duarte, A. R., and Souza, G. L. (2018). Multi-objective performance improvements of general finite single-server queueing networks. *Journal of Heuristics*, *24*(5), 757-781.

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

Demir, L., Tunali, S., and Løkketangen, A. (2011). A tabu search approach for buffer allocation in production lines with unreliable machines. *Engineering Optimization*, *43*(2), 213-231.

Demir, L., Tunalı, S., and Eliiyi, D. T. (2012). An adaptive tabu search approach for buffer allocation problem in unreliable non-homogenous production lines. *Computers and Operations Research*, *39*(7), 1477-1486.

Diamantidis, A. C., and Papadopoulos, C. T. (2004). A dynamic programming algorithm for the buffer allocation problem in homogeneous asymptotically reliable serial production lines. *Mathematical problems in Engineering*, *2004*(3), 209-223.

Diamantidis, A. C., and Papadopoulos, C. T. (2009). Exact analysis of a two-workstation one-buffer flow line with parallel unreliable machines. *European Journal of Operational Research*, *197*(2), 572-580.

Diamantidis, A., Lee, J.-H., Papadopoulos, C. T., Li, J., and Heavey, C. (2020). Performance evaluation of flow lines with non-identical and unreliable parallel machines and finite buffers. *International Journal of Production Research*, *58*(13), 3881-3904.

Dolgui, A., Eremeev, A., Kolokolov, A., and Sigaev, V. (2002). A genetic algorithm for the allocation of buffer storage capacities in a production line with unreliable machines. *Journal of Mathematical Modelling and Algorithms*, *1*(2), 89-104.

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

Dolgui, A., Eremeev, A. V., and Sigaev, V. S. (2007). HBBA : Hybrid algorithm for buffer allocation in tandem production lines. *Journal of Intelligent Manufacturing*, *18*(3), 411-420.

Gershwin, S. B., and Schor, J. E. (2000). Efficient algorithms for buffer space allocation. *Annals of Operations Research*, *93*(1), 117-144.

Helber, S., Schimmelpfeng, K., Stolletz, R., and Lagershausen, S. (2011). Using linear programming to analyze and optimize stochastic flow lines. *Annals of Operations Research*, *182*(1), 193-211.

Hillier, F. S., and SO, K. C. (1991). The effect of the coefficient of variation of operation times on the allocation of storage space in production line systems. *IIE transactions*, *23*(2), 198-206.

Ho, Y.-C., Eyler, M. A., and Chien, T.-T. (1979). A gradient technique for general buffer storage design in a production line. *International Journal of Production Research*, *17*(6), 557-580.

Huang, M.-G., Chang, P.-L., and Chou, Y.-C. (2002). Buffer allocation in flow-shop-type production systems with general arrival and service patterns. *Computers and Operations Research*, *29*(2), 103-121.

Kassoul, K., Belhaouari, S. B., and Cheikhrouhou, N. (2021). Dynamic Cognitive-Social Particle Swarm Optimization. *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*, 200-205.

Kassoul, K., Cheikhrouhou, N., and Zufferey, N. (2022a). Buffer allocation design for unreliable production lines using genetic algorithm and finite perturbation analysis. *International Journal of Production Research*, *60*(10), 3001–3017.

Kassoul, K., Zufferey, N., Cheikhrouhou, N., and Belhaouari, S. B. (2022b). Exponential Particle Swarm Optimization for Global Optimization. *IEEE Access*,10, 78320-78344. doi: 10.1109/ACCESS.2022.3193396.

Kennedy, J., and Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95-International Conference on Neural Networks*, *4*, 1942-1948.

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

Li, J., and Meerkov, S. M. (2009). Mathematical Modeling of Production Systems. In J. Li and S. M. Meerkov (Eds.), *Production Systems Engineering* (p. 1-59). Springer US.

Liberopoulos, G. (2018). Performance evaluation of a production line operated under an echelon buffer policy. *IISE Transactions*, *50*(3), 161-177.

Lin, J. T., and Chiu, C.-C. (2018). A hybrid particle swarm optimization with local search for stochastic resource allocation problem. *Journal of Intelligent Manufacturing*, *29*(3), 481-495.

MacGregor Smith, J., and Cruz, F. R. (2005). The buffer allocation problem for general finite buffer queueing networks. *IIE Transactions*, *37*(4), 343-365.

Massim, Y., Yalaoui, F., Amodeo, L., Chatelet, E., and Zeblah, A. (2010). Efficient combined immune-decomposition algorithm for optimal buffer allocation in production lines for throughput and profit maximization. *Computers and Operations Research*, *37*(4), 611-620.

Nahas, N., Ait-Kadi, D., and Nourelfath, M. (2006). A new approach for buffer allocation in unreliable production lines. *International Journal of Production Economics*, *103*(2), 873-881.

Narasimhamu, K. L., Reddy, V. V., and Rao, C. S. P. (2014). Optimal Buffer Allocation in Tandem Closed Queuing Network with Single Server Using PSO. *Procedia Materials Science*, *5*, 2084-2089.

Narasimhamu, K. L., Reddy, V. V., and Rao, C. S. P. (2015). Optimization of Buffer Allocation in Manufacturing System Using Particle Swarm Optimization. *International Review on Modelling and Simulations (IREMOS)*, *8*(2), 212-222.

Papadopoulos, C. T., O'Kelly, M. E. J., and Tsadiras, A. K. (2013). A DSS for the buffer allocation of production lines based on a comparative evaluation of a set of search algorithms. *International Journal of Production Research*, *51*(14), 4175-4199.

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

Respen, J., Zufferey, N., and Amaldi, E. (2016). Metaheuristics for a job scheduling problem with smoothing costs relevant for the car industry. *Networks*, *67*(3), 246-261.

Respen, J., Zufferey, N., and Wieser, Ph. (2017). Three-level inventory deployment for a luxury watch company facing various perturbations. Jou*rnal of the Operational Research Society*, *68*(10), 1195-1210.

Sabuncuoglu, I., Erel, E., and Gocgun, Y. (2006). Analysis of serial production lines : Characterisation study and a new heuristic procedure for optimal buffer allocation. *International Journal of Production Research*, *44*(13), 2499-2523.

Schindl, D., and Zufferey, N. (2015). A Learning Tabu Search for a Truck Allocation Problem with Linear and Nonlinear Cost Components. *Naval Research Logistics*, 61(1), 32-45.

Shi, C., and Gershwin, S. B. (2016). A segmentation approach for solving buffer allocation problems in large production systems. *International Journal of Production Research*, *54*(20), 6121-6141.

Shi, L., and Men, S. (2003). Optimal buffer allocation in production lines. *IIE Transactions*, *35*(1), 1-10.

Spinellis, D. D., and Papadopoulos, C. T. (2000). A simulated annealing approach for buffer allocation in reliable production lines. *Annals of Operations Research*, *93*(1), 373-384. Spinellis, D., Papadopoulos, C., and Smith, J. M. (2000). Large production line optimization using simulated annealing. *International journal of Production Research*, *38*(3), 509-541.

Thevenin, S., and Zufferey, N. (2019). Learning Variable Neighborhood Search for a Scheduling Problem with Time Windows and Rejections, *Discrete Applied Mathematics*, *261*, 344 – 353.

Weiss, S., Schwarz, J. A., and Stolletz, R. (2019). The buffer allocation problem in production lines : Formulations, solution methods, and instances. *IISE Transactions*, *51*(5), 456-485.

Xi, S., Chen, Q., Smith, J.M.G., Mao, N., Yu, A., and Zhang, H. 2020. "A new method for solving buffer allocation problem in large unbalanced production lines." *International Journal of Production Research*, 58(22), 6846-6867.

Yamashita, H., and Altiok, T. (1998). Buffer capacity allocation for a desired throughput in production lines. *IIE Transactions*, *30*(10), 883-892.

Yegul, M. F., Erenay, F. S., Striepe, S., and Yavuz, M. (2017). Improving configuration of complex production lines via simulation-based optimization. *Computers and Industrial Engineering*, *109*, 295-312.

Zhang, H.-Y., Chen, Q.-X., Smith, J. M., Mao, N., Yu, A.-L., and Li, Z.-T. (2017). Performance analysis of open general queuing networks with blocking and feedback. *International Journal of Production Research*, *55*(19), 5760-5781.

Zhao, C., Li, J., Huang, N., and Horst, J. A. (2016). Flexible serial lines with setups : Analysis, improvement, and application. *IEEE Robotics and Automation Letters*, *2*(1), 120-127.

Zhou, W., and Lian, Z. (2011). A tandem network with a sharing buffer. *Applied Mathematical Modelling*, *35*(9), 4507-4515.

Zufferey, N. (2012). Optimization by ant algorithms: possible roles for an individual ant. *Optimization Letters*, *6(5)*, 963-973.

Zufferey, N. (2016). Tabu Search Approaches for Two Car sequencing problems with smoothing constraints. In *Metaheuristics for Production Systems* (p. 167-190). Springer.

Chapter 3

Buffer allocation design for unreliable production lines using Genetic Algorithm and Finite Perturbation Analysis

3.1 Introduction

The design of unreliable manufacturing systems, such as production lines with human operators or automated assembly systems, has received considerable attention from both academics and industrial worlds. Numerous researches are dedicated to the *Buffer Allocation Problem* (BAP), that is, how much buffer capacities to allow and where to place them within the line (Dolgui et al. 2007). A classification review of the studies on BAPs is presented in (Weiss et al. 2018) and in (Demir et al. 2014).

Most of the methods for designing production lines are based on flow simulations that require numerous iterations (Shi and Men 2003). In the industry, it is difficult to implement such methods since running them requires a huge amount of time. Therefore, this study focuses on reducing the time needed to reach convergence, while trying to reach the optimal buffer allocation scheme. An efficient technique that shows good convergence features is the *Finite Perturbation Analysis* (FPA) method. This method is an analytic technique that provides a solution to the design problem after a single simulation run of the system's model (Suri and Leung, 1989). This chapter proposes a hybrid method using the *Genetic Algorithm* (GA) and the FPA approaches, coupled to discrete-event simulation technique. The advantage of using GA lies in the fact that it offers a strong diversification ability to explore the solution space. GA is coupled here with FPA to allow an intensification of the search in some regions of the solution space. An exploration direction in the searched area is determined by calculating the expectation of the gradient of the performance measure (in this chapter, the performance measure is the production rate) with respect to the system parameters. The gradient is used in an allocation technique of buffer capacities for unreliable production lines operating in a stochastic environment. The advantage of the technique is that it converges in a single simulation run and ensures scalability, as it can address different sizes of production lines (with different numbers of machines and buffers). To the best of our knowledge, combining

GA with FPA has never been addressed in the literature, which shows the novelty of our work.

The chapter is organized as follows. Section 3.2 gives the formulation of the problem and reviews the related work of BAP in production lines. In Section 3.3, we present the overall approach for solving the BAP. Section 3.4 gives a detailed development of the used optimization techniques. Section 3.5 presents the experiments, the results and their discussions. Finally, conclusions and future research directions are provided in Section 3.6.

3.2 Presentation of the problem and related work

In this section, we first explain the BAP in serial production lines and we formulate the problem. Second, we provide a comprehensive literature review.

3.2.1 Problem description and formulation

Consider a serial production line that consists of a sequence $(M_1, M_2, \ldots, M_n)$ of n machines connected in series. The machines are separated by $(n-1)$ buffers $(b_1, b_2, \ldots, b_{n-1})$, where b_i represents the size of the buffer located between the machines M_i and M_{i+1}, B_{max} is a fixed non-negative integer that represents the total buffer space available for the whole production line, and $f(s)$ is a mathematical function that represents the *Production Rate* (PR) of the production line (see Figure 3.1). Each part enters the system through the first machine, passes sequentially through all machines and the intermediate buffers, and then exits the line through the last machine. In open production lines, the first machine is never starved (i.e., there are always available parts at the input of the system), and the last machine is never blocked (i.e., there is available space for parts storage at the output of the system). It is assumed that the transfer times to move parts from a machine to a buffer and vice versa are negligible.

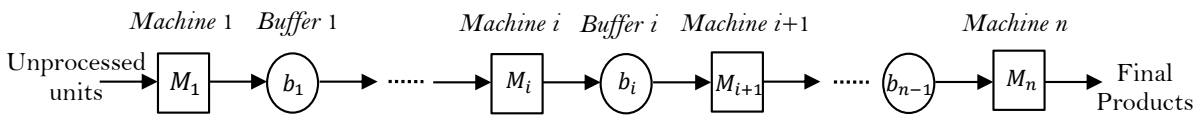


Figure 3.1. Open serial production line

Formulation: The design problem considered in this chapter consists in allocating B_{max} throughout $(n-1)$ buffers. The objective is to maximize the average production rate in open serial production lines. In mathematical terms, the problem can be written as follows:

$$\text{Find:} \quad s = (b_1, b_2, \ldots, b_{n-1}) \quad \text{so as to maximize} \quad f(s) \tag{3.1}$$

$$\text{Subject to:} \quad \sum_{i=1}^{n-1} b_i = B_{max}; \quad b_i \geq 0 \text{ and integer (for each } i) \tag{3.2}$$

3.2.2 Literature review

The design of production lines is a well-known problem in industry. It can be separated into two main categories. The first category focuses on designing the total cycle time across workstations, whereas the second one (namely, the BAP) focuses on buffer allocation between machines to store work in process. Regarding the latter, several authors address this problem for small-sized production lines by using analytical models and techniques. Markov chain models are one of the analytical techniques that have been widely addressed in the literature. To reach the optimal buffer allocation, Hillier and So (1991) use an exact analytic model based on Markov-chain representations for small-sized production lines. However, they show that any attempt to expand the model to a realistic situation leads to state space problems. Diamantidis and Papadopoulos (2009) study a serial flow line with two workstations and an intermediate buffer. They propose an analytical approach based on a Markov process model to describe the characteristics of the system. For more details on timed Markov models of manufacturing systems, the reader can refer to (Papadopoulos et al. 2019).

To evaluate large systems, approximation methods are developed relying on decomposition or aggregation methods. These methods are applied if a huge state space makes the numerical solution of Markov chains too slow or impossible (Weiss et al. 2018). The decomposition method (Liberopoulos 2018; Diamantidis et al. 2019; Xi et al. 2020) is based on a decomposition of the *n*-machine line into a set of $(n-1)$ two-machine lines where the buffer between the upstream machine and the downstream machine has the same capacity. On the other hand, the aggregation methods (Zhao et al. 2017) replace a two-machine line with an equivalent single machine, and then reduce iteratively the number of machines until obtaining only one machine. This reduction allows for a fast performance evaluation of long lines. Recently, some researchers use metaheuristics to solve BAPs for reliable or unreliable machines. For details about metaheuristics, the reader can find some principles and rules in (Zufferey 2012; Gendreau and Potvin 2019). Nahas et al. (2006) propose a new local search approach based on the degraded ceiling metaheuristic to solve the BAP in unreliable

production lines. They conclude that the degraded ceiling gives better results in an acceptable amount of time, and as the size of the line increases, the solutions become more competitive. Spinellis and Papadopoulos (2000) apply two stochastic methods, GA and simulated annealing, for solving the BAP with the objective of maximizing the production rate. They show that both methods can be used for optimizing large production lines. Spinellis, Papadopoulos, and MacGregor (2000) present a generalized queuing network algorithm as an evaluative procedure for optimizing production-line configurations using simulated annealing. Dolgui et al. (2002) propose an approximate method based on the Markov-model aggregation approach using a GA. They obtain better solutions in comparison with other methods. Costa et al. (2015) propose a new parallel tabu-search algorithm using an adaptive neighborhood generation mechanism, which consists in minimizing the total buffer capacity of a serial production system under a minimum *PR* constraint. In another study, Vergara and Kim (2009) develop a buffer allocation method that can be applied to open or closed production lines. They use a heuristic to find a near-optimal solution and the performance of the heuristic is compared to the performance of GA for various test cases. Demir et al. (2011) use an adaptive tabu-search algorithm where the *PR* of the production line is evaluated by using a decomposition method. Recently, Cruz et al. (2018) develop a multi-objective GA to allocate the buffers and service rates to generate a set of solutions for more than one objective functions. Yegul et al. (2017) focus on the problem of optimizing production-line configurations and propose several simulation-based optimization approaches based on ant-colony optimization. Kose and Kilincci (2018) combine two metaheuristics and propose a hybrid-evolutionary simulation-optimization approach for the multi-objective BAP in open serial production lines.

In other studies, authors propose to hybridize metaheuristics with other search techniques. Dolgui et al. (2007) present a hybrid optimization algorithm, using GA and branch-and-bound approaches. For the evaluation of objective function, they propose a Markov-model aggregation technique. Bierlaire et al. (2010) have combined variable neighborhood search (Thevenin and Zufferey 2019) (with the aim to explore the solution space) with a modified trust-region algorithm (for intensification purposes), for nonlinear global optimization. For solving the BAP in unreliable machines, Shi and Men (2003) propose a tabu search (Schindl and Zufferey 2015) to speed up the nested partitions method. Amiri and Mohtashami (2011) propose a GA integrated to line-search method for determining the optimal/near-optimal size of each buffer storage. According to Demir et al. (2014), only 4 out of 95 studies after 1998 employ a hybrid approach to optimize the buffer sizes, and just one of those researches solves

the problem in a multi-objective manner whereas the others solve the BAP while maximizing the *PR*. The reader can refer to Li et al. (2009) for more details on the *PR* evaluation. They review and summarize the most important publications and methods dealing with the *PR* analysis of production systems with unreliable machines and finite buffer capacities.

With respect to the objective function, the majority of studies done in the literature deal with a single objective problem. Some papers have addressed a bi-objective problem formulation (Lee et al. 2009; Amiri and Mohtashami 2011; Cruz et al. 2012), and other studies involve three objectives (Gershwin and Schor 2000; Cruz et al. 2010; Zandieh 2017). This observation is also true for the job-scheduling field, where only a few studies have considered jointly more than two objective-function components (e.g., Thevenin et al. 2016; Thevenin et al. 2017).

Simulation-optimization approaches are often employed for production lines. The simulation allows evaluating, at each execution, the stationary performance measure, whereas the optimization algorithm provides a direction for searching new solutions for the design of the system. Battini et al. (2009) use a simulation approach to determine the optimal buffer allocation for a serial production line that increases the reliability of the production line by minimizing the micro-downtime of machines. For their simulation, they consider the impact of the cost on the buffer allocation. Can and Heavey (2011) employ a genetic programming to perform symbolic regression. They develop a sampling approach adapted to genetic programming, obtaining simulation-based meta-models of industrial systems. Gürkan (2000) optimizes the *PR* with respect to buffer capacities using sample-path optimization to find optimal buffer allocations by working with a continuous-line approach instead of discrete lines. This is because continuous-line simulations are substantially faster and the approximations are quite accurate. Dealing with continuous parameters enables him to compute directional derivatives of the *PR* that are valuable for optimization purposes using infinitesimal perturbation analysis (Turki et al. 2013). Cheikhrouhou et al. (2002) propose an algorithm designed to determine simultaneously the sizes of storage space and service times in stochastic transfer lines. Based on FPA, the developed technique retrieves information on throughput gradient with respect to system parameters. The provided information is integrated into a multidimensional optimization procedure to determine the best configuration of the system parameters.

Classical simulation-optimization methods and techniques based on evolutionary algorithms (Fu 1994) consume significant computing time. Indeed, they require a large number of simulations before an optimal/near-optimal solution can be found. Therefore, this work uses

FPA that provides a solution to the design problem using a single long simulation of the system, significantly reducing the computation time. The advantage of FPA is to be able to use the same simulation to estimate the gradients and to develop the configuration using a stochastic algorithm. Indeed, during the simulation, it is possible to obtain updated values of the gradient of the considered performance measure at regular time intervals. At each update, these values are used to guide the search for solutions throughout the simulation.

Although hybridizing different methods has been proposed to solve the BAP, to the best of our knowledge, we could not find any study combining GA and FPA to solve the BAP in unreliable production lines for maximizing the production rate. The proposed approach exploits the advantages of the two algorithms. GA tends to quickly identify promising regions of the solution space, whereas FPA deeply explores such regions to find competitive solutions. Moreover, an experimental study is carried out to identify the best parameters of the proposed method. The algorithms are applied equally to short and large production lines. A contribution of our work is thus the development of a general optimization approach (GA combined with FPA) using a single simulation for the design of buffer capabilities for stochastic production lines. Another advantage of this approach is that it can take into account different sizes of production system.

3.3 Problem solving framework

The proposed approach is based on a combination of GA with FPA. Figure 3.2 represents the detailed optimization approach, where the same FPA algorithm forms the core element providing the different gradients estimates for the *Stochastic Algorithm* (SA). The advantage of using GA is to approach competitive solutions in a reasonable time. The interest of FPA is to use, on the one hand, the same simulation to estimate the gradients, and on the other hand, to develop the configuration with a SA. In other words, the optimization and the simulation take place simultaneously. In the initial population of m different solutions, $P = (s_1, s_2, ..., s_m)$, each solution $s_i = (b_{i,1}, b_{i,2}, ..., b_{i,j}, ..., b_{i,n-1})$ is generated randomly from a discrete uniform distribution over nonnegative integers, where $b_{i,j}$ denotes the j^{th} buffer of the i^{th} solution. The evolution of the current population P to the new population $P' = (s'_1, s'_2, ..., s'_m)$ is achieved by the application of the selection and crossover operators for a number of generations, with the objective of approaching the zone containing an optimal/near-optimal solution $s^* = (b_1^*, b_2^*, ..., b_{n-1}^*)$. Each solution (or configuration) of P' is the input of the SA based on the calculation of the gradients of the production rate $\partial f / \partial bi$ with respect to the decision

variables (buffer capacities) $b_i, i = 1, 2,..., n-1$. This solution method is derived from the optimization approaches by the technique of gradient descent (Robbins and Monro 1951).

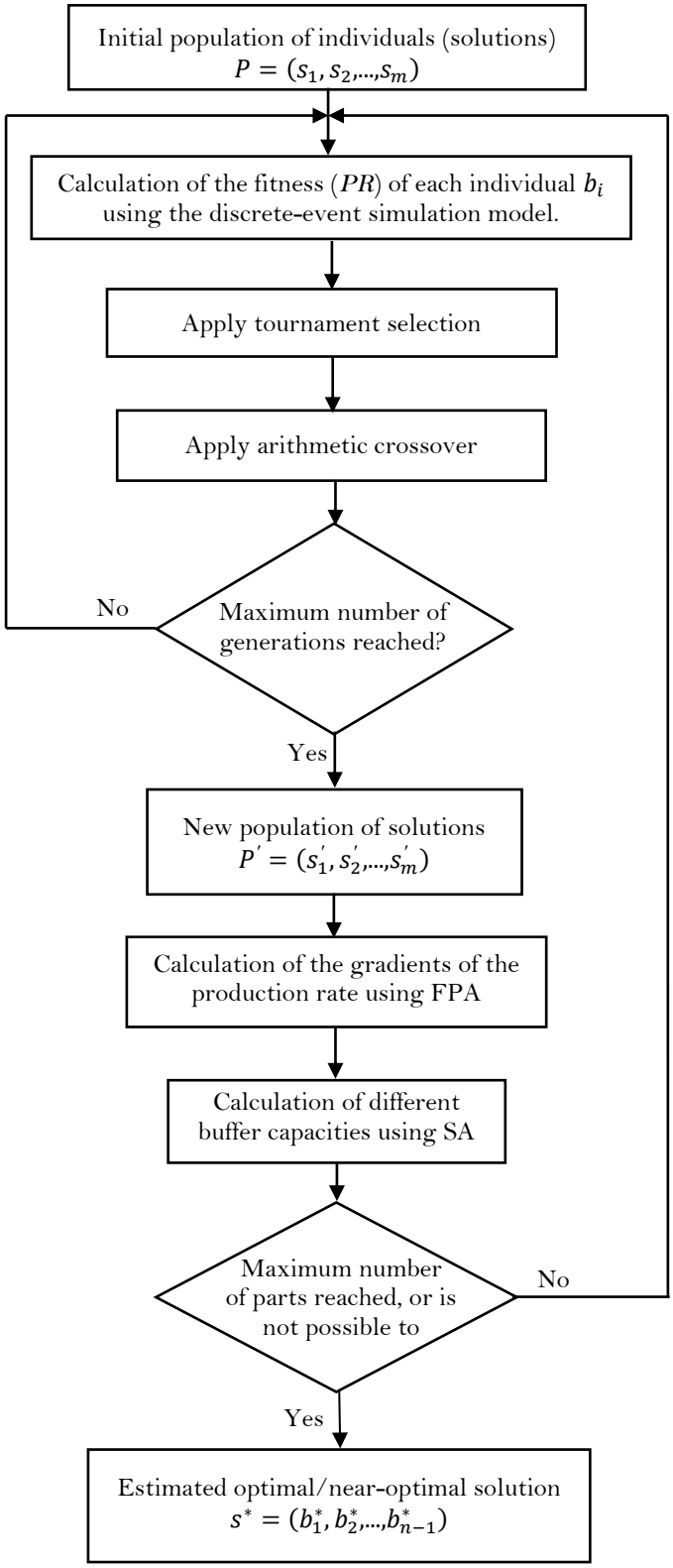


Figure 3.2. Framework of our solution approach for the BAP

Consider the original simulation that consists in simulating P parts. Each iteration k is done by simulating L parts (where $L < P$), after which a new evaluation of the gradients is calculated by FPA. This evaluation determines the new search direction for the design solution. This search method requires, for each iteration k, the estimation of the projections' vectors of the gradients on the hyperplane $\sum_{i=1}^{n-1} b_i = B_{max}$. Indeed, for a parameter b_i, the projected gradients allow to determine the best direction for improving the current solution (Robbins and Monro 1951). Thereafter, an iterative hill-climbing-type procedure is used. The objective is to evolve from the solution found towards a better solution using the gradient descent technique. The iteration $(k+1)$ improves, from one configuration to another, the parameters b_i^k, representing the buffer capacity b_i at the iteration k.
Hence,

$$b_i^{k+1} = b_i^k + a_k \cdot \left(\frac{\partial f}{\partial b_i} - \frac{1}{n-1}\sum_{i=1}^{n-1}\frac{\partial f}{\partial b_i}\right) \quad (3.3)$$

where,

- a_k is a numeric suite that verify the following conditions of convergence:

$$\lim_{k\to\infty} a_k = 0 \quad , \quad \sum_{k=1}^{\infty} a_k = \infty, \quad \sum_{k=1}^{\infty} a_k^2 < \infty \quad (3.4)$$

- The term $\left(\frac{\partial f}{\partial b_i} - \frac{1}{n-1}\sum_{i=1}^{n-1}\frac{\partial f}{\partial b_i}\right)$ in equation (3.3) represents the projection of the gradient $\frac{\partial f}{\partial b_i}$ on the hyperplane constraints $\sum_{i=1}^{n-1} b_i = B_{max}$.

SA stops if it is not possible to improve the PR anymore with a new solution (the gradients' values become negligible), or when the maximum number of parts produced in the simulation is reached.

3.4 Development of the optimization technique

3.4.1 Genetic algorithm (GA)

GA is a global optimization technique that considers an optimization problem as the environment where feasible solutions are the individuals living in that environment. A set of feasible solutions takes the place of a population of organisms. Every organism represents a possible valid solution to the problem. In general, a GA begins with a randomly generated set of individuals. Once it has been generated, the GA enters a main loop. After each iteration

(generation), the population changes and a new population is produced by applying a number of stochastic operators to the previous population (Tomassini 1999).

We present below the components of the proposed GA, relying on the following notation.

- N is the number of individuals
- $P(t)$ is the current population
- $P(t+1)$ is the new population
- *MinBuffer* is the smallest value allowed
- *MaxBuffer* is the largest value allowed
- *PositionBuffer* is the rank of the buffer
- *GapStock* is a parameter that allows adjusting the difference between *MinBuffer* and *MaxBuffer*
- $U\{a, b\}$ is a function that randomly generates (with a uniform distribution) an integer in the set {a, b}
- $Int[c]$ is an operator that takes only the integer value of c

Initial population: the first step of the proposed GA consists of building an initial population using Algorithm 3.1. The initial population is composed of m different solutions (or configurations). Each configuration is generated randomly with the discrete uniform distribution. To satisfy Constraint (2), some buffer values are then increased or decreased in a well-distributed manner i.e., a repair procedure is performed to make the solution feasible. The repair procedure works as follows. It distributes, over the buffers, the value of a possible gap between (a) and (b), where (a) is the sum of the buffer capacities that are generated by the uniform distribution, and (b) is the total capacity B_{max} required by Constraint (2) ($\sum_{i=1}^{n-1} b_i = B_{max}$). If the gap is negative as (a) < (b) (resp. positive as (a) > (b)), the repair is performed by sequentially increasing (resp. decreasing) by one unit the value of some b_i's, picked up randomly among the b_i's that are below (resp. above) the largest (resp. smallest) b_i, until a feasible solution is reached). To illustrate this procedure, we give an example of allocation of 30 buffers at four positions. Suppose that the uniform distribution gives the configuration (3, 8, 6, 10), which amounts to 27 (the gap is thus equal to –3). Therefore, to satisfy the constraint of reaching 30, the repair procedure could increase by one unit the values of the first three stocks to finally give the following configuration (4, 9, 7, 10). The main objective of this phase is to have configurations that cover, in a non-biased manner, various regions of the solution space.

Algorithm 3.1. Generation of initial population of *m* individuals

While the number of individuals is not reached, do

For each individual, **do**

- SumBuffer $= 0$ *(initialization of the sum of values of all the capacities allocated to the stocks)*
- MinBuffer $=\text{Int}[B_{max}/(n-1)] - \text{GapStock}$ *($n-1$ denotes the size of the generated individual; n is the number of machines)*
- MaxBuffer $= \text{Int}[B_{max}/(n-1)] + \text{GapStock}$

For each position, **do**

- PositionBuffer $\sim$ U{MinBufer,MaxBuffer} (*position gives the rank of the buffer*)
- PositionBuffer = SumBuffer + PositionBuffer

$D = B_{max} -$ PositionBuffer (*if* $D \neq 0$, *the repair procedure is applied*)

- If $D = 0$, the individual is already feasible
- If $D > 0$, increase in a well-distributed way some values of the individual
- If $D < 0$, reduce in a well-distributed way some values of the individual

Evaluation: the fitness value of each individual, namely f, represents the *PR* of the production line and is the result of the execution of a simulation model (Arena).

Solution encoding*:* each decision variable is associated with a gene. A set of genes constitutes a chromosome, and one or several chromosomes represent an individual (i.e., a solution). In this chapter, there is one type of gene: buffer capacity. Thus, each individual consists of a unique chromosome $s = (b_1, b_2, ..., b_{n-1})$.

Selection: the tournament selection is used, which is a useful and robust selection mechanism commonly used by GAs (Miller and Goldberg 1995). It consists in first selecting randomly two individuals of the population, then in comparing their *PR*, and finally in selecting the winner for the next generation.

Crossover: two parent solutions of the current population interact and give two child solutions (offsprings) of the next population (for the next generation). The major reason that makes this crossover operator very suitable to the problem under consideration is that it assures feasibility of the offspring. Since both parents are feasible, both children must also be feasible (Matondang and Jambak 2010). The crossover phenomenon is essential because it allows GA to explore the search space efficiently. In this chapter, we use the arithmetic crossover operator with a constraint criterion (Duman 2018). From a population of parents, two parent solutions $s_1$and s_2 are first selected. Next, the two new offsprings s_1' and s_2' are generated by linear combination of s_1 and s_2 as follows:

$$s_1' = \alpha \cdot s_1 + (1 - \alpha) \cdot s_2 \quad (3.5)$$

$$s_2' = (1 - \alpha) \cdot s_1 + \alpha \cdot s_2 \quad (3.6)$$

where α is a random weighting coefficient picked uniformly in the interval [0,1]. Note that in order to respect Constraint (3.2), a balanced adjustment for the arithmetic crossover is used. More precisely, some values of the offsprings are increased or decreased in a well-distributed manner in order to respect the total buffer space available for the whole production line (i.e., the repair procedure is performed). Moreover, since the buffer sizes must be integer, some values of the offsprings are modified to satisfy this condition. The former repair procedure and the latter rounding mechanism correspond here to the mutation operator.

The proposed GA is described in Algorithm 3.2.

Algorithm 3.2. Genetic Algorithm

Initialization

- Set $t = 0$
- Generate randomly an initial population $P(t)$ of m individuals
- Calculate the production rate of each individual of $P(t)$ using the Arena simulation model

While $P(t+1)$ does not contain N individuals, **do**

- Repeat $N/2$ times the below process:
  Select randomly two parents from $P(t)$ and apply the tournament selection based on the fitness f
  Clone the best-selected individual
- Apply arithmetic crossover operator between two parent solutions s_1 and s_2
- Insert the offsprings s_1' and s_2' in the new population $P(t+1)$

Repeat the While loop as long as an overall stopping condition is not met

3.4.2 Finite perturbation analysis (FPA)

Discrete-Event Dynamic Systems (DEDS), such as manufacturing systems, are dynamic asynchronous system where the state transitions are initiated by the occurrence of discrete events in the system at instants of time. The evolution in time of a DEDS is analyzed using its trajectory, denoted by *Nominal Trajectory* (NT), representing the events that affect the entities processed by the system. Assuming that a parameter of the system is perturbed in a trajectory following the occurrence of an event, the value of the parameter changes, and a new trajectory is obtained, denoted by *Perturbed Trajectory* (PT).

Perturbation analysis (PA) is an analytical technique that calculates the performance-measure sensitivity of a DEDS with respect to system parameters, by analyzing its sample path (Ho 1987). PA allows obtaining the perturbed performance from a nominal (original) experiment or sample path without the need of doing a perturbed experiment. When the perturbations are finite (large amplitude), and where the changes of order of events in the trajectory do not influence the performance measure of the system, an extension of the PA must be introduced. It is called FPA or first order PA (Cao 1987). We can thus emit the hypothesis that once a perturbation is introduced, the type of future interactions that it can encounter and/or lead through system (machines, buffers and parts in the system) in the PT is statistically similar to that in the nominal one (HO 1987). Hence, the construction and simulation of the PT is not necessary and only the NT will be used to define the system's response to a perturbation. This development is supported by specific experimental results of particular DEDS (Heidelberg et al. 1988).

The purpose of applying the perturbation analysis is to provide a prediction of an estimate of *PR* through the development of two sets of rules: the perturbation generation rules and the perturbation propagation. These rules are based on the prediction of future events and their respective durations after the introduction of perturbations in the NT by changing the capacity of some buffers. First, we consider the unitary cell shown in Figure 3.3 for the development of the FPA rules. Next, from the different results obtained, we can generalize them to the entire production line.

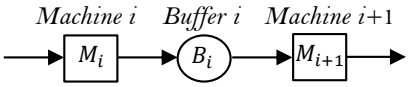


Figure 3.3. Unitary cell

Consider the sequence of events of the NT of the machine M_i, denoted as $M_i(t)$, and that of M_{i+1}, denoted as $M_{i+1}(t)$, and the buffer level $b_i(t)$ of the Unitary cell (see Figure 3.4). The occurrence of a long-duration failure of M_{i+1} leads to an accumulation of parts in buffer b_i, supplied by M_i that is in operating order. At time t_1, $b_i(t)$ reaches its maximum capacity B_i and any part produced by M_i cannot be stored in the buffer. Although machine M_i remains operational, it becomes blocked (noted FO) until the instant t_2, when the machine M_{i+1} is finally repaired and becomes operational. Next, it begins to process parts from the intermediate stock b_i. Hence, this marks the end of the blocked period for machine M_i. At time

t_3, M_i breaks down, allowing M_{i+1} to continue to process the parts already in the stock b_i. This explains the reduction in the level of b_i, which subsequently continues to vary with the different states of M_i and M_{i+1}. The blocked period of Figure 3.4 could be reduced if the buffer capacity of b_i is higher.

During the period $[t_1, t_1]$, the storage space remains saturated, as shown on the trajectory. This is due to our choice of representation of the total stock capacity in the model that takes into account the unit on M_i waiting to be placed in b_i. This unit is directly stored as soon as a place is released in b_i. Moreover, since the buffer is a discrete parameter, we should have represented the evolution of the level of b_i in steps, but for sake of clarity, we preferred a representation where the buffer level is considered as if it is a continuous variable.

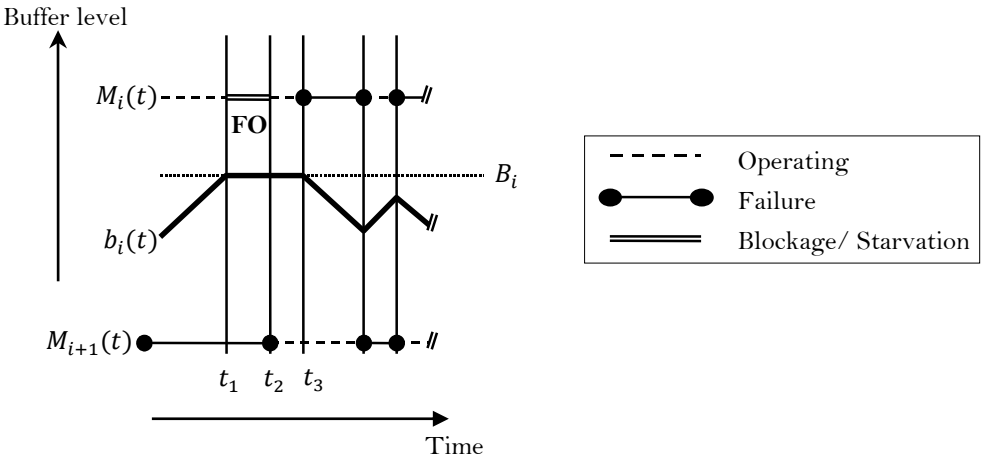


Figure 3.4. Nominal trajectory of a unitary cell

3.4.2.1 Perturbation generation rules in a unitary cell

All the propositions hereafter are described in detail in (Cheikhrouhou, 2001) and they form the core of the FPA.

Proposition 1: Let $M_i(t)$ be a blocked machine (FO) during a period $[t_1, t_2]$ of the nominal trajectory. A perturbation (or local gain) of size Δt_i is created in the sequence of events of this machine, if a variation in the size of stock Δb_i is injected when M_i passes through the FO period (or earlier). The perturbed trajectory of M_i, denoted $M_i'(t)$, is specified as follows:

$$M_i'(t) = M_i(t + \Delta t_i), \forall t > t_1 \quad (3.7)$$

$$\Delta t_i = t_i \cdot \Delta b_i \quad (3.8)$$

where t_i is the average service time of M_i, and Δt_i is the resultant perturbation in its trajectory.

Figure 3.5a graphically reflects equations (3.7) and (3.8) for an eligible perturbation provoked by $\Delta b_i = 1$ and where the sequence of events $M_i(t)$ is translated in time by a value of Δt_i corresponding to the equivalent gain in production time.

Note that if the perturbation is not eligible for this interval, i.e., $(t_1 - t_2) < \Delta t_i$, the FO period would be completely removed from the perturbed trajectory. The sequence of events of M_i would be advanced by a period equal to $(t_1 - t_2)$. Moreover, the perturbation may be considered as a local gain because it represents a potential gain in the global execution time, if it is passed on to the global duration of the production.

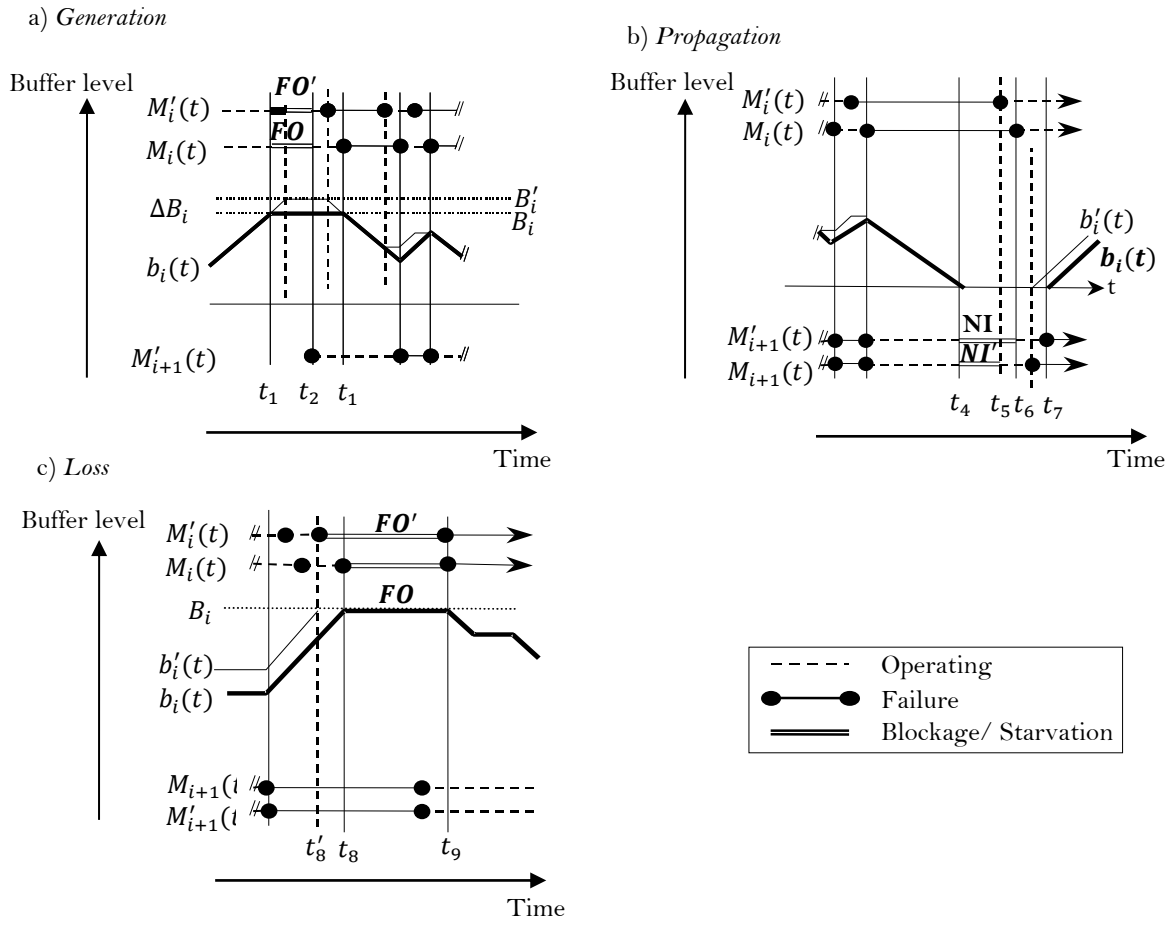


Figure 3.5. Generation, propagation and loss of perturbation in a unitary cell

3.4.2.2 Perturbation propagation rules in a unitary cell

Proposition 2: Let $[t_4, t_6]$ be a time interval in which machine M_{i+1} of the unitary cell is starved (noted NI). The operation $\Delta b_i = 1$ does not change the input date of period t_4 between the nominal trajectory and the perturbed trajectory.

Hence,

$$b_i(t_4) = b'_i(t_4) = 0 \tag{3.9}$$

where $b'_i(t)$ is the buffer level in the PT. The same result can be formulated in the most general case with eligible perturbations $\Delta b_i = p, \forall p \in \mathbb{N}$.

Proposition 3: Equation (3.7) and the hypothesis of the existence of a period of famine (NI) (due to a failure for example) for M_{i+1} during the following period $[t_4, t_6]$ in the NT induces a reduction of this interval in the PT. The new time interval NI is $[t_4, t_5]$, calculated from (3.10). The perturbation is therefore completely propagated from M_i to M_{i+1}.

$$NI'[t_4, t_5] = FO[t_4, t_6] - \Delta t_i \tag{3.10}$$

$$M'_{i+1}(t) = M_{i+1}(t + \Delta t_i), \quad \forall t > t_5 \tag{3.11}$$

The sequence of events of M_i denoted by $M'_i(t)$ in its perturbed trajectory M_{i+1} is represented in figure 3.5b. The value of the total propagated perturbation from M_i to M_{i+1} is calculated by:

$$\Delta t'_i = t_6 - t_5 = \Delta t_i \tag{3.12}$$

Proposition 4: Suppose that the trajectory $M_i(t)$ is translated to the left by Δt_i, due to a particular gain (propagation of perturbation) at an earlier time (in the perturbed trajectory). Since M_{i+1} did not undergo any perturbation until that moment and B_i has not been modified, the occurrence of an event FO during $[t_8, t_9]$ in the nominal trajectory of $M_i(t)$ implies an increase of the FO period in the perturbed trajectory and consequently, the loss of the perturbation (see Figure 3.5c.):

$$FO'[t_8, t_9] = FO[t_8, t_9] + \Delta t_i \tag{3.13}$$

$$M'_i(t) = M_i(t), \forall t > t_8 \tag{3.14}$$

Note that if the duration of the blockage is considered very short, the same scheme is applicable with the same explanation. However, this period may disappear in the perturbed trajectory if the value of the perturbation is longer than the duration of the blockage.

3.4.3 Description of the FPA and SA algorithms

3.4.3.1 Notation

P	number of parts for each simulation
L	number of parts for each iteration
T	duration of a simulation
f	production rate in the nominal trajectory ($f = P/T$)
f^*	estimated production rate in the perturbed trajectory
$\partial f / \partial bi$	gradients of the production rate due to buffer decision variable b_i
b_i^k	buffer between the machine i and the machine $i+1$ iteration k
Sum_i	accumulation of the gain time on machine i
A	regulation parameter for the convergence
Anint	an Arena operator that takes the near integer value
Abs	absolute value
Arg min	an arena operator that takes the minimum value

We assume that the simulation of a production line required to produce P parts is sufficiently long so that all possible events can occur during the simulation run. The PR is estimated by f, where T is the duration of the simulation. Then for a given i,

$$\frac{\partial f}{\partial b_i} = \frac{\partial (P/T)}{\partial b_i} = -\frac{f}{T} \cdot \frac{\partial T}{\partial b_i} \tag{3.15}$$

where $\partial T / \partial bi$ represents the total perturbation time affecting the line, due to the introduction of the perturbation Δb_i in the simulation. The value of $\partial T / \partial bi$ is calculated by applying the generation and propagation rules of the FPA mentioned above.

3.4.3.2 FPA algorithm

The presented FPA relies on (Suri 1989) and it is applicable only for DEDS flow-shop such as manufacturing systems or communication networks. FPA includes the detection of critical events in the nominal trajectory (such as the FO and the NI), and their respective durations. Therefore, we can predict the repercussion of a perturbation of *one unit capacity* of the stock associated with each relevant event. This repercussion concerns the persistence of the event

in the perturbed trajectory and its duration. Algorithm 3.3 describes the FPA algorithm for the optimization of buffer capacities. It works simultaneously with the simulation of the nominal trajectory in order to detect in time the events which make it possible to generate and propagate the perturbations, such as the beginning of a FO period or the end of a NI period.

Algorithm 3.3. Finite Perturbation Analysis

Generation of perturbations

Step 1

- **For** $i = n$ to $n-1$, **do**: $Sum_{i+1} = 0$; Select b_i (*initialization of accumulators* Sum_{i+1})

Step 2

- **For** « $M_{i+1}(t)$ is FO for the first time », **do**: $B_i = B_i + \Delta B_i$; $Sum_i = t_i \cdot \Delta B_i$

Propagation of perturbation

Step 3

- **For** any event « $M_i(t)$ is NI », **do**: $Sum_i = max\{Sum_j + [NI], Sum_{i+1}\} - max\{0, [NI]\}$

Step 4

- **For** any event « $M_{i+1}(t)$ is FO », **do**: $Sum_i = max\{Sum_{i+1} + [FO], Sum_i\} - max\{0, [FO]\}$

Step 5

If $i = n$ (Last machine) **then**

- $\frac{\partial f}{\partial b_i} = -\left(\frac{f}{T}\right) \cdot Sum_n$
- $f^* = \frac{P}{T - Sum_n}$; Stop (f^* *denotes the estimated production rate in the perturbed trajectory*)

Else Go to Step 1

3.4.3.3 Stochastic Algorithm (SA)

The proposed SA is based on the Robbins-Monro procedure. It is an iterative procedure used to update the buffer capacities during the simulation run, i.e., if at each update a new calculation of the gradients (thus a launch of the FPA algorithm) is done, we need as many launches as the number of buffer updates. The total duration of simulation would be too long. To overcome this disadvantage, the solution is to develop a single simulation optimization (mono-run). FPA is then integrated into SA (see Algorithm 3.4) that is used to reach a potential optimal/near-optimal solution. It is based on the nominal trajectory that is available to update the configuration settings (here, the buffer capacities) for each production of a number L of parts. Step 3 presents the modified procedure of Robbins-Monro that uses the

technique of gradient calculation based on FPA. A perturbation is indeed an incrementation of one unit of the capacity of a buffer that shows blockage events during his nominal trajectory. This incrementation is virtual and used only to evaluate the Production Rate gradient thanks to the propagation rules of FPA. Actually, the values of the buffer capacities are updated during the optimization using SA and not through their incrementation by one unit. Moreover, each update of the buffer capacities at an iteration $(k+1)$ respects the main constraint $\sum_{i=1}^{n-1} b_i^{k+1} = \sum_{i=1}^{n-1} b_i^k = B_{max}$,$\forall\, n$, which leads to the fact that, at any iteration, the algorithm keeps the allocation of the same global capacity on all line buffers. Step 4 is a precaution case where some updated values of b_i^{k+1} are negatives. We choose a_k as numeric suites of type A/k , where A is a constant randomly determined by choosing an initial value. The adjustment is done through the calculation of the projected gradient values, so that the step size between the iterations is consistent. In other words, the value of b_i after each iteration is in the same order of magnitude.

Algorithm 3.4. Stochastic Algorithm

Initialization $k = 1$

Step 1

- Choose the initial values of b_i^k, for $i = 1$ to $n-1$

Step 2

- Simulate for b_i^k L parts $(L < P)$
- Estimate $\frac{\partial f}{\partial b_i}$ by using FPA

Step 3

- $b_i^{k+1} = b_i^k + \left(\frac{A}{N}\right) \cdot \left(\frac{\partial f}{\partial b_i} - \frac{1}{n-1}\sum_{i=1}^{n} \frac{\partial f}{\partial b_i}\right)$

Step 4

If $b_i^{k+1} \leq 0$ **then**

- $b_p^k = Arg\ min\ b_i^k$; $b_p^{k+1} = Arg\ min\ b_i^{k+1}$
- $a \sim U\,(0,1)$; $f = \left|\frac{a \cdot b_p^k}{b_p^{k+1} - b_p^k}\right|$
- $b_i^{k+1} = b_i^k + f \cdot \left(\frac{A}{N}\right) \cdot \left(\frac{\partial f}{\partial b_i} - \frac{1}{n-1}\sum_{i=1}^{n} \frac{\partial f}{\partial b_i}\right)$

Step 5

If $\left|b_p^{k+1} - b_p^k\right| \leq \varepsilon$ **then**

- $b_i = Anint\left(b_i^k\right)$; Stop

If P parts are simulated **then** Stop

Else $k = k + 1$; go to step 2

Since we are dealing with discrete values, a discretization of stock values would be necessary. The idea is to consider the capacities of the stocks as continuous variables until the end of simulation of L parts. After that, the algorithm transforms the value of the parameter to a discrete value (Step 5). The parameter ε represents a trade-off between the simulation length and the accuracy of the estimated parameter, for which a value of 0.0001 is taken in this work. The advantage of this stopping criterion is that it takes into account the variations of the all the b_i variables simultaneously. Therefore, the algorithm stops when the difference between the values of b_i from an update to another becomes negligible or when the maximum number of parts produced in the simulation is reached.
Note that by changing the values of b_i during the single-run simulation, transient effects are introduced into the system and the *PR* values and gradients thus contain biases related to these effects. The algorithm proposed at this stage is therefore considered as a heuristic.

3.5 Numerical Experiments

In Subsection 5.1, the parameters of the proposed method are identified. Numerical comparisons of our methods are performed in Subsection 3.5.2 with respect to the literature. Four sets of instances are considered, involving 3, 5, 10 and 20 machines in the production line. Finally, Section 3.5.3 demonstrates the benefit of combining GA with FPA when compared to GA or FPA only. In each table, the best results are always highlighted in bold face. The *Arena* simulation language V14.0 is used to develop the models (Altiok and Melamed 2010) and the algorithms are implemented in Java. The experiments are performed on a PC with a 1.9-GHz Core (TM) i5CPU processor with 8 GB of RAM.

Termination criterion: the algorithm stops when the production rate obtained at a given generation is lower than the best production rate obtained so far (only valid after the 10^{th} generation), or when the maximum number of 20 generations is reached.

3.5.1 Determination of the parameters of the methods

On the one hand, we have to determine the number R of replications. A large R means a long computational time, and conversely a simulation with a small R can lead to a biased solution. It is therefore necessary to determine when the method converges into fewer replications. Several test cases for different sizes of production lines are conducted to predict the evolution of the average *PR* with respect to R.

Figure 3.6 shows four instances involving 3, 5, 10 and 20 machines, with R ranging from 1 to 50. It contains: (a) a Tukey's boxplot to reflect the distributions of the *PRs*; (b) a graph of the average *PR* for different values of R. From the graphs, we can see that the average *PR* is stable from R = 30 and above. This coincides with the law of large numbers, which states that the average of a large independent random variable approximately converges to their expectation. Generally, this approximation can be used when the size of the sample is greater than 30. Therefore, in this chapter, R = 30 replications will be used.

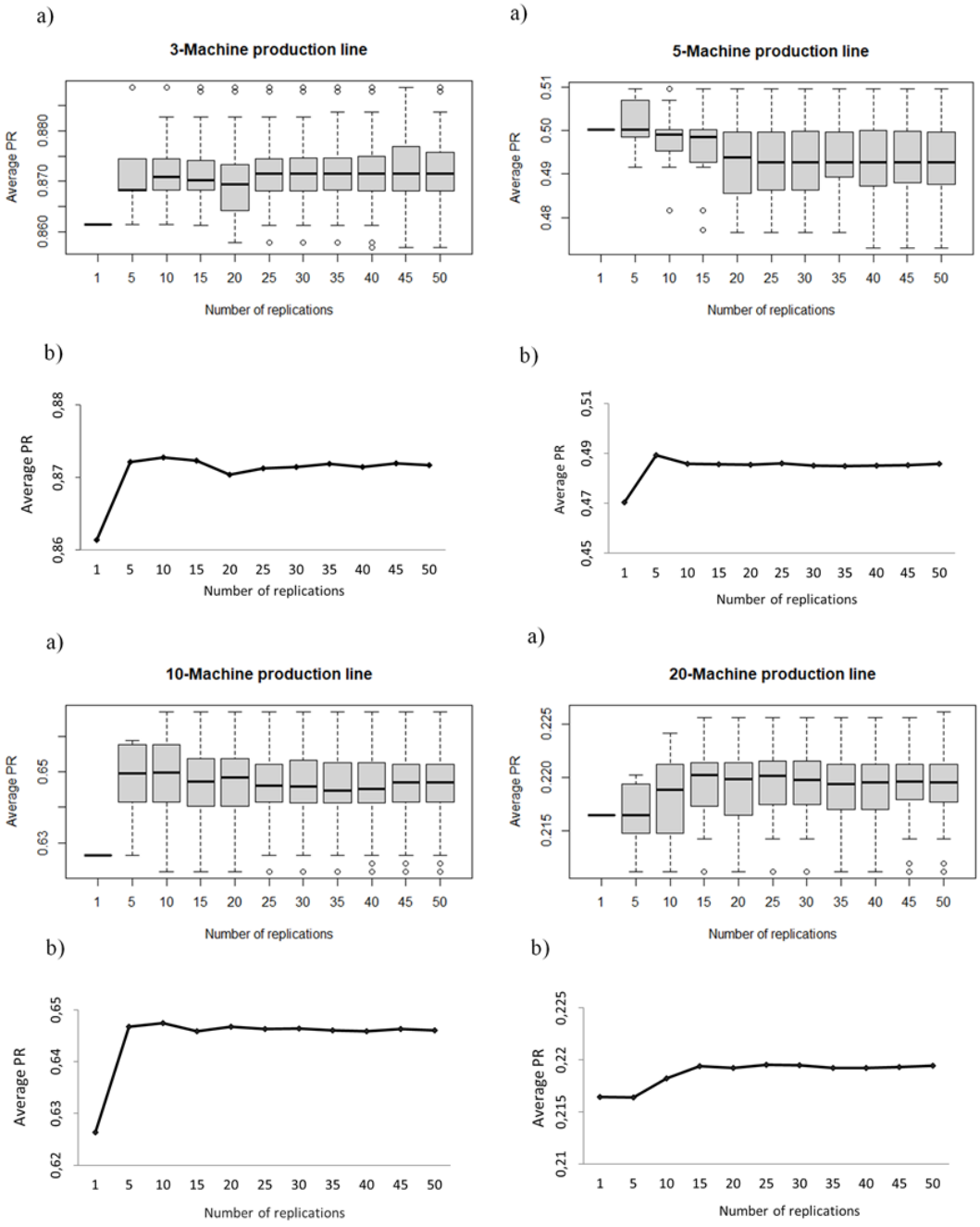


Figure 3.6. Evolution of production rate with different numbers of replications

On the other hand, we have to determine the operator parameters of GA, as they play a crucial role in the convergence of the algorithm. Hence, for a given initial population, the convergence can be obtained faster while obtaining the best possible solution if the operators are well designed (Pavai and Geetha 2019). After preliminary experiments, we have set 20 generations and 30 individuals for GA. For the selection operator, we have tested *Random Selection* (RS) and *Tournament Selection* (TS). For the crossover operator, we have tested *Single-Point Crossover* (SPC) and *Arithmetic Crossover* (AC). The computational results are summarized in Table 3.1. The first two columns in this table depict the size of the production line (i.e., the number of machines and the total buffer size available). The next columns show the average *PR* for the following combinations of operators (RS-SPC), (RS-AC), (TS-SPC), and (TS-AC), respectively. One can see that for the case with 3 machines, all the PR values are equal. However, when the number of machines increases, the best results are obtained when using the (TS-AC) combination.

Table 3.1. Identification of parameters of GA

Number of machines	B_{max}	**Average *PR***			
		RS / SPC	RS / AC	TS / SPC	TS / AC
3	20	0.871783	0.871783	0.871783	**0.871783**
5	31	0.494708	0.494708	0.494789	**0.496413**
10	270	0.647766	0.646903	0.645671	**0.649204**
20	100	0.351846	0.354975	0.354220	**0.358180**
40	200	0.436278	0.438015	0.436319	**0.438583**

3.5.2 Comparison of the proposed method with respect to the literature

GA-FPA is compared with different methods from the literature (see Table 3.2). All the runs are conducted under the same experimental conditions. More precisely, in all sets of problems, it is assumed that the service times of each machine is one time unit, the machines are subject to breakdown, and the failure and repair times follow a geometric distribution with a probability of p_i and r_i, respectively (these values will be presented later).

Table 3.2. State-of-the-art methods for buffer allocation

Authors	Objective	System /method	Proposed method
Ho et al. (1979)	Max *PR*	Simulation	Gradient technique based on perturbation analysis
Gershwin and Schor (2000)	Max PR Min (work in process, total buffer size)	Decomposition	Gradient search
Shi and Men (2003)	Max PR	Decomposition	Tabu search and nested partitions
Nahas et al. (2006)	Max PR	Decomposition	Degraded ceiling algorithm and simulated annealing
Demir et al. (2011)	Max *PR* Min (work in process, total buffer size)	Decomposition	Adaptive tabu search
Massim et al. (2010)	Max (*PR*, Profit)	Decomposition	Artificial immune algorithm
Kose and Kilincci (2015)	Max *PR*	Simulation	Genetic algorithm and simulated annealing

3.5.1.1 Results for a 3-machine production line

Consider a production line composed of 3 machines and 2 buffers. The information on the machines are given in Table 3.3. This example is proposed by Gershwin and Schor (2000), and used as a comparison benchmark by Massim et al. (2010). In their study, Gershwin and Schor (2000) do not mention explicitly the buffer configuration found and they only give an idea on how production rate varies with respect to buffer sizes. This is the reason why GA-FPA is compared only with the value proposed by Massim et al. (2010). The total buffer capacities to be allocated among the line is 20. Table 3.3 shows that GA-FPA obtains higher *PR*.

Table 3.3. Parameters and results obtained for a 3-machine production line

Machine	**1**	**2**	**3**
r_i	0.35	0.15	0.4
p_i	0.037	0.015	0.02
Authors	B_{max}	**Buffer size configuration**	**Average *PR***
Massim et al. (2010)	20	14 6	0.86799
GA-FPA	20	13 7	**0.87178**

3.5.1.2 Results for a 5-machine production line

This example was initially proposed by Ho et al. (1979) and used in different papers as a benchmarking case (Gershwin and Schor 2000; Massim et al. 2010; Demir et al. 2011; Kose and Kilincci 2015). The total buffer size is 31, and the parameters of the five machines are presented in Table 3.4. For this example, the average production rate is estimated by simulating 100,000 parts with 50 replications. MTTR and MTBF denote the mean time to repair and the mean time between failures, respectively. Table 3.4 presents the buffer sizes configuration and the corresponding average production rates obtained by the different approaches. GA-FPA obtains a slightly better production rate with the buffer size configuration (7, 11, 9, 4), which has the same allocation at the extremities of the line as Kose and Kilincci (2015), Demir et al. (2011), Massim et al. (2010) and Gershwin and Schor (2000). The allocation of small capacities at the beginning and at the end of the line, and important sizes in the middle, is likely to be efficient for facilitating the passage of the parts and thus bypassing a possible congestion of the line. Again, we can see that GA-FPA obtains the best results.

Table 3.4. Parameters and results obtained for a 5-machine production line.

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

Authors	B_{max}	**Buffer size configuration**	**Average *PR***
Ho et al. (1979)	31	5 11 8 7	0.4914
Gershwin and Schor (2000)	31	7 10 10 4	0.4943
Demir et al. (2011)	31	7 10 10 4	0.4943
Massim et al. (2010)	31	7 10 10 4	0.4943
Kose and Kilincci (2015)	31	7 10 10 4	0.4943
GA-FPA	31	7 11 9 4	**0.4948**

3.5.1.3 Results for a 10-machine production line

This instance is proposed by Nahas et al. (2006) and used in (Massim et al. 2010; Demir et al. 2011; Kose and Kilincci 2015). The dataset is shown in Table 3.5. The total buffer size to be allocated is 270. As shown in Table 3.5, Demir et al. (2011) and Nahas et al. (2006) obtain the same buffer configuration (14, 19, 30, 54, 45, 27, 23, 24, 34) with an average *PR* of 0.64135.

Massim et al. (2010) report a slightly better solution of 0.64139 with an almost similar distribution of buffers (14, 19, 30, 52, 47, 27, 23, 24, 34). GA-FPA generates the best *PR* with a different buffer allocation (19, 23, 24, 45, 43, 34, 22, 29, 31). It can be noted that when the machines are balanced, all the methods generate, in most cases, distributions of buffers capacities with more storage areas allocated in the middle of the line. This buffer allocation is likely to avoid blockages and to facilitate parts flowing downstream.

Table 3.5. Parameters and results obtained for a 10-machine production line

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 = $1/p_i$	20	3	22	22	25	40	23	30	45	20

Authors	B_{max}	**Buffer size configuration**									**Average *PR***
Nahas et al. (2006)	270	14	19	30	54	45	27	23	24	34	0.64135
Demir et al. (2011)	270	14	19	30	54	45	27	23	24	34	0.64135
Massim et al. (2010)	270	14	19	30	52	47	27	23	24	34	0.64139
Kose and Kilincci (2015)	270	7	16	48	61	24	41	20	34	19	0.63016
GA-FPA	270	19	23	24	45	43	34	22	29	31	**0.64920**

3.5.1.4 Results for a 20-machine production line

The three first columns in Table 3.6 present the parameters of the production line. The fourth and fifth columns give the results obtained by Demir et al. (2011) and Kose and Kilincci (2015), respectively. The last column gives the results obtained by GA-FPA, which outperforms the other methods for 8 out of 9 instances. The case where our method does not outperform the existing methods is due to the lower value of the Production Rate of one of the replications where the PR value is 0.20357. In this case, the nominal trajectory in the simulation presents a high number of starving periods (NI) which increased the total production time to produce the needed number of parts, thus decreasing the production rate.

Table 3.6. Results obtained for a 20-machine production line

Parameters			**Average *PR***		
p_i	r_i	B_{max}	Demir et al. (2011)	Kose and Kilincci (2015)	GA-FPA
0.1	0.1	100	**0.234191**	0.226499	0.223559
0.2	0.2	100	0.297025	0.302448	**0.305203**
0.3	0.3	100	0.334450	0.349517	**0.354811**
0.4	0.4	100	0.359637	0.354299	**0.394316**
0.5	0.5	100	0.377895	0.382401	**0.422175**
0.6	0.6	100	0.391846	0.397120	**0.443916**
0.7	0.7	100	0.402760	0.446904	**0.461407**
0.8	0.8	100	0.411638	0.450032	**0.476649**
0.9	0.9	100	0.419018	0.459111	**0.487760**

3.5.3 Benefit of combining GA and FPA

In order to measure the benefit of combining GA with FPA in a single method GA-FPA, an idea is to compare GA-FPA with its components GA and FPA considered alone. Results are given for instances with 5, 10 and 20 machines, in Tables 3.7, 3.8 and 3.9, respectively. In each table, the data is first given (column 1 to 3), followed by the average *PR* for GA, FPA and GA-FPA (column 4 to 6), and finally, computing times (to reach the best encountered solutions) are provided in seconds at the end (column 7 to 9). The instances are characterized as follows: the service time of each machine is one time unit; the failure and repair times follow a geometric distribution with a probability of p_i and r_i, respectively; all machines are identical and have the same failure and repair probabilities, ranging from $p_i = r_i = 0.1$ to $p_i = r_i = 0.9$ with increments of 0.1. The total buffer size to be allocated is 50, 100 and 200 for the instances with 5, 10 and 20 machines, respectively.

On the one hand, we can easily observe that regarding quality, GA-FPA obtains the best *PA* results for each of the 27 instances (note that for two instances with 5 machines, GA obtains equivalent solutions). On the other hand, regarding speed, GA-FPA is the quickest method for 13 out of 27 instances. It seems that for the larger instances with 20 machines, GA-FPA need more time to converge to its best solutions. The success of GA-FPA can be explained by the fact that the GA component can quickly identify promising regions of the solutions space (exploration ability), whereas the FPA component can intensify the search in such regions (exploitation ability).

Table 3.7. Parameters and results for a 5-machine production line

Parameters			**Average *PR***			**CPU (s)**		
p_i	r_i	B_{max}	GA	FPA	GA-FPA	GA	FPA	GA-FPA
0.1	0.1	50	0.344618	0.34141	**0.34471**	797	***363***	468
0.2	0.2	50	**0.406667**	0.40539	**0.40666**	235	881	***226***
0.3	0.3	50	0.437662	0.43667	**0.43803**	475	377	***320***
0.4	0.4	50	0.457296	0.45633	**0.45754**	581	387	***251***
0.5	0.5	50	0.469631	0.46880	**0.47016**	686	***338***	455
0.6	0.6	50	0.479729	0.47906	**0.47999**	**744**	845	762
0.7	0.7	50	0.486405	0.48586	**0.48644**	597	624	***359***
0.8	0.8	50	0.491948	0.49163	**0.49215**	784	753	***588***
0.9	0.9	50	**0.495934**	0.49581	**0.49593**	367	***287***	310

Table 3.8. Parameters and results for a 10-machine production line

Parameters			Average *PR*			CPU (s)		
p_i	r_i	B_{max}	GA	FPA	GA-FPA	GA	FPA	GA-FPA
0.1	0.1	100	0.33326	0.33034	**0.33568**	1435	2337	**1122**
0.2	0.2	100	0.42294	0.41603	**0.42357**	1899	1841	**1177**
0.3	0.3	100	0.47545	0.46883	**0.47553**	2708	2067	**1941**
0.4	0.4	100	0.50570	0.50210	**0.50640**	1563	**1486**	1645
0.5	0.5	100	0.52402	0.52080	**0.52476**	1573	1304	**1035**
0.6	0.6	100	0.53311	0.52743	**0.53387**	1559	1731	**1140**
0.7	0.7	100	0.53135	0.52810	**0.53240**	1603	**1243**	1270
0.8	0.8	100	0.52493	0.52417	**0.52617**	1051	2528	**950**
0.9	0.9	100	0.51585	0.51469	**0.51629**	1940	**1333**	1368

Table 3.9. Parameters and results for a 20-machine line

Parameters			Average *PR*			CPU (s)		
p_i	r_i	B_{max}	GA	FPA	GA-FPA	GA	FPA	GA-FPA
0.1	0.1	200	0.28679	0.27509	**0.28692**	2559	4586	**2293**
0.2	0.2	200	0.36492	0.35213	**0.36765**	4880	**2295**	3032
0.3	0.3	200	0.40889	0.39919	**0.40902**	3261	**1890**	2287
0.4	0.4	200	0.43590	0.43002	**0.43604**	3881	**3577**	3950
0.5	0.5	200	0.45352	0.44696	**0.45361**	**4430**	6049	4620
0.6	0.6	200	0.46687	0.45923	**0.46726**	3616	**2105**	2340
0.7	0.7	200	0.47695	0.47320	**0.47712**	**2212**	6861	3489
0.8	0.8	200	0.48488	0.48372	**0.48494**	3934	3601	**3295**
0.9	0.9	200	0.49140	0.49143	**0.49144**	**1367**	5287	1930

3.5.4 Sensitivity analysis

When all machines of the production line are identical (the same service times and the same probabilities of failure and repair), all the methods (our method as well as the methods with which we made a comparison) generate, in most cases, allocation of buffers capacities with more storage areas for the middle of the line. The purpose of the sensitivity analysis proposed is to determine whether this allocation is still respected in the case when the production line is unbalanced (different service times and parameters) and to determine, otherwise, whether there is a specific buffer allocation pattern

Three series of experiments were carried out and a description of each of the test cases is presented. In all cases, the PR of the buffer allocations obtained by the GA-FPA approach is estimated by simulating 10,000 parts for 30 replications.

3.5.1.5 *Experiment set 1*

In this case, we consider a 5-machine unbalanced production line. The total buffer size is set to 30. The different service times of the five machines are presented in the first column of Table 3.10, where the service time of one machine is, in each case, bigger than the other machines. The machines are subject to breakdown, and the failure and repair times follow geometric distributions with $p_i = 1/300$, and $r_i = 1/10$ respectively.

Table 3.10. Parameters and results for set of experiments 1 for a 5-machine line

Case	Service times	Buffer size configuration			
1	$t_1 = 3, t_2 = t_3 = t_4 = t_5 = 1$	10	9	6	5
2	$t_2 = 3, t_1 = t_3 = t_4 = t_5 = 1$	7	11	7	5
3	$t_3 = 3, t_1 = t_2 = t_4 = t_5 = 1$	6	9	11	4
4	$t_4 = 3, t_1 = t_2 = t_3 = t_5 = 1$	4	9	7	10
5	$t_5 = 3, t_1 = t_2 = t_3 = t_4 = 1$	6	6	8	10

3.5.1.6 *Experiment set 2*

We examine in this case the buffer allocation in a 5-machine balanced production line. It is assumed that the service time of every machine is one time unit, the failure times follow a geometric distribution with a probability $p_i = 1/200$. The repair rates of the five machines are different, and presented in the first column of Table 3.11. In each of the 5 cases, just one machine needs less time to repair compared to the others machines.

Table 3.11. Parameters and results for set of experiments 2 for a 5-machine line

Case	Repair rates	Buffer size configuration			
1	$1/r_1 = 30, 1/r_2 = 1/r_3 = 1/r_4 = 1/r_5 = 10$	5	7	12	6
2	$1/r_2 = 30, 1/r_1 = 1/r_3 = 1/r_4 = 1/r_5 = 10$	6	8	10	6
3	$1/r_3 = 30, 1/r_1 = 1/r_2 = 1/r_4 = 1/r_5 = 10$	5	9	10	6
4	$1/r_4 = 30, 1/r_1 = 1/r_2 = 1/r_3 = 1/r_5 = 10$	7	11	6	6
5	$1/r_5 = 30, 1/r_1 = 1/r_2 = 1/r_3 = 1/r_4 = 10$	7	11	8	4

3.5.1.7 *Experiment set 3*

This test examines the buffer allocation in a 5-machine balanced production line. The service times of each machine is one time unit and the repair rates of all machines are set to $r_i = 1/10$. The different failure rates of the five machines are presented in the first column of Table 3.12. In each case, just one machine has a high probability of breaking down compared to the other machines.

Table 3.12. Parameters and results for set of experiments 3 for a 5-machine line

Case	Failure rates	Buffer size configuration			
1	$1/p_1 = 100, 1/p_2 = 1/p_3 = 1/p_4 = 1/p_5 = 300$	10	9	6	5
2	$1/p_2 = 100, 1/p_1 = 1/p_3 = 1/p_4 = 1/p_5 = 300$	7	11	7	5
3	$1/p_3 = 100, 1/p_1 = 1/p_2 = 1/p_4 = 1/p_5 = 300$	6	9	11	4
4	$1/p_4 = 100, 1/p_1 = 1/p_2 = 1/p_3 = 1/p_5 = 300$	4	9	7	10
5	$1/p_5 = 100, 1/p_1 = 1/p_2 = 1/p_3 = 1/p_4 = 300$	6	6	8	10

The results of the experiment set 1 (see Table 3.10) show that a larger stock size should be allocated to the input stock of the slower machine. This would allow the fastest upstream machines to continue to be operational and thus avoid any possible blockage.

The results of the experiment set 2 (see Table 3.11) show that in the case where the repair rates are different, the allocation found is comparable with the one in the case of balanced production lines, where a high accumulation of intermediate stocks is required. This buffer allocation is likely to avoid blockages and to facilitate parts flowing downstream.

From the results of the experiment set 3 (see Table 3.12), a configuration tends to persist, where the stock after the machine with a larger MTBF would require larger storage capacities. This could be explained by the need to facilitate parts flowing along the line, and thus prevent the upstream machines from being blocked and the downstream machines from starving.

3.6 Conclusion

This chapter proposes a quick and efficient hybrid simulation-optimization approach combining the genetic algorithm (GA) and the finite perturbation analysis (FPA) for the buffer allocation problem (BAP) of serial production lines with unreliable machines (the capacity of each buffer has to be decided, without exceeding the total available capacity for the production line). The heart of the proposed method is a stochastic algorithm based on the gradient estimates by FPA, complemented by GA for finding its input solutions. The role of GA is to approach quickly the optimal/near-optimal solution-space regions (thanks to its diversification and exploration ability), whereas the role of FPA is to investigate deeply such regions to find better solutions (thanks to its intensification and exploitation ability). An advantage of the proposed approach is to use a single simulation only. This is achieved because FPA can evaluate gradients of steady production rates at regular intervals during the

simulation. The performed experiments show on the one hand that GA-FPA outperforms the state-of-the-art methods from the literature on instances where the number of machines varies from 3 to 20 machines. On the other hand, GA-FPA outperforms both GA solely and FPA solely with respect to solution quality, and often with respect to speed (i.e., the time needed to find its best solutions).

Various research directions could be derived from this work. First, GA-FPA could be extended to other types of production systems with different features, such as assembly/disassembly systems or transportation systems. Next, due to advances in manufacturing technologies and the rise of Industry 4.0, new solutions are opening up to improve the efficiencies and the production rates of these systems. The emergence of intelligent and interconnected collaborative industrial robots and storage sensors allows to monitoring the inventory levels and the different operations achieved by the robots, including material transfer between the different robots and operators in the workshop. This opens an avenue of research in the design of production lines. In fact, designing production lines would be not only allocating storage capacities on the different buffers, but it extends to the allocation of service times over the different robots and humans that are in charge of executing the operations on the line. Our design approach, based on optimization using simulation, can be interesting to tackle such problems. In that case, the GA-FPA algorithms will be used to allocate, simultaneously, buffer capacities and robot/operator service times in a single simulation-optimization run.

3.7 References

Altiok, T. and B. Melamed. 2010. "Simulation modeling and analysis with Arena". *Elsevier.*

Amiri, M., and A. Mohtashami. 2011. "Buffer allocation in unreliable production lines based on design of experiments, simulation, and genetic algorithm." *International Journal of Advanced Manufacturing Technology* 62 (1-4): 371-383.

Battini, D., A. Persona, and A. Regattieri. 2009. "Buffer size design linked to reliability performance: A simulative study." *Computers and Industrial Engineering 56*(4): 1633-1641.

Bierlaire, M., M. Thémans, and N. Zufferey. 2010. "A heuristic for nonlinear global optimization." *INFORMS Journal on Computing* 22(1): 59-70.

Can, B., and C. Heavey. 2011. "Comparison of experimental designs for simulation-based symbolic regression of manufacturing systems." *Computers and Industrial Engineering* 61(3): 447-462.

Cao, X. 1987. "First order perturbation analysis of a simple multiclass finite source queue." *Performance Evaluation* 7: 31-41.

Cheikhrouhou, N, J-L. Paris and H. Pierreval. 2002. "Une méthode de dimensionnement de lignes de transfert via l'analyse de perturbation finie." *Journal Européen des Systèmes Automatisés* 36 (2): 199-221.

Cheikhrouhou, N. 2001. "Apport de l'Analyse de Perturbation pour le Dimensionnement et le Pilotage des Systèmes de Production." *PhD thesis*, *Grenoble Institute of Technology*, https://www.theses.fr/2001INPG0001.

Costa, A., A. Alfieri, A., Matta, and S. Fichera. 2015. "A parallel tabu search for solving the primal buffer allocation problem in serial production systems." *Computers and Operations Research* 64: 97-112.

Cruz, F.R.B., A.R. Duarte, and G.L. Souza. 2018. "Multi-objective Performance Improvements of General Finite Single-Server Queueing Networks." *Journal of Heuristics* 24 (5): 757-781.

Cruz, F.R.B., G.Kendall, L.While, A.R. Duarte, and N.L.C. Brito. 2012. "Throughput maximization of queueing networks with simultaneous minimization of service rates and buffers." *Mathematical Problems in Engineering*, 1-19.

Cruz, F.R.B., T. Van Woensel, and J.M. Smith. 2010. "Buffer and throughput trade-offs in M/G/1/K queuing networks: A bicriteria approach." *International Journal of Production Economics* 125: 224-234.

Demir, L., S. Tunali, and D.T. Eliiyi. 2014. "The state of the art on buffer allocation problem: a comprehensive survey." *Journal of Intelligent Manufacturing* 25(3): 371-92.

Demir, L., S. Tunali, and D.T. Eliiyi. 2011. "An adaptive tabu search approach for buffer allocation problem in unreliable non-homogenous production lines." *Computers andOperations Research* 39(7): 1477-1486.

Diamantidis, A., J-H Lee, C.T. Papadopoulos, J. Li, and C. Heavey. 2019. "Performance Evaluation of Flow Lines with Non-identical and Unreliable Parallel Machines and Finite Buffers." *International Journal of Production Research*, 1-24.

Diamantidis, A., and C. T. Papadopoulos. 2009. "Exact Analysis of a Two-workstation One-buffer Flow Line with Parallel Unreliable Machines." *European Journal of Operational Research* 197 (2): 572-580.

Dolgui, A., A. Eremeev, and V. Sigaev. 2007. "HBBA: Hybrid algorithm for buffer allocation in tandem production lines." *Journal of Intelligent Manufacturing* 18(3): 411- 420.

Dolgui, A., A. Eremeev, A. Kolokolov, and V. Sigaev. 2002. "A genetic algorithm for the allocation of buffer storage capacities in a production line with unreliable machines." *Journal of Mathematical Modeling and Algorithms* 1: 89-104.

Duman, S. 2018. "A Modified Moth Swarm Algorithm Based on an Arithmetic Crossover for Constrained Optimization and Optimal Power Flow Problems." *IEEE Access* (6): 45394- 45416.

Fu, M.C. 1994. "Optimization via simulation: A review." *Annals of Operations Research* 53: 199-248.

Gendreau, M., and J.-Y. Potvin. 2019. "Handbook of Metaheuristics." *International Series in Operations Research and Management Science*. Springer International Publishing.

Gershwin, S.B., and J.E. Schor. 2000. "Efficient algorithms for buffer space allocation." *Annals of Operations Research* 93: 117-144.

Gürkan, G. 2000. "Simulation optimization of buffer allocations in production lines with unreliable machines." *Annals of Operations Research* 93: 177-216.

Heidelberg, P., X. Cao, M. Zananis, and R. Suri. 1988. "Convergence properties of infinitesimal perturbation analysis estimates." *Management Science* 34 (11): 1281-1302.

Hillier F.S., and K.C. So. 1991. "The effect of machine breakdowns and interstage storage on the performance of production line systems." *International Journal of Production Research* 29 (10): 2043-2055.

Ho, Y.-C. 1987. "Performance evaluation and perturbation analysis of discrete event dynamic systems." *IEEE Transactions on Automatic Control* 32: 563-572.

Ho, Y.C., M.A. Eyler, and T.T. Chien. 1979. "A gradient technique for general buffer storage design in a production line." *International Journal of Production Research* 17(6): 557-580.

Kose, S.Y., and O. Kilincci. 2018. "A Multi-Objective Hybrid Evolutionary Approach for Buffer Allocation in Open Serial Production Lines." *Journal of Intelligent Manufacturing*, 1-19.

Kose, S.Y., andO. Kilincci. 2015. "Hybrid approach for buffer allocation in open serial production lines." *Computers and Operations Research* 60: 67-78.

Lee, H-T., S.K. Chen, and S. Shunder. 2009. "A meta-heuristic approach to buffer allocation in production line." *Journal of Creative Communication and Innovative Technology* 38(1): 167-178.

Li, J., D.E. Blumenfeld, N. Huang, and J.M Alden. 2009. "Throughput analysis of production systems: Recent advances and future topics." *International Journal of Production Research* 47(14): 3823-2851.

Liberopoulos, G. 2018. "Performance Evaluation of a Production Line Operated Under an Echelon Buffer Policy." IISE Transactions 50 (3): 161-177.

Massim Y., F. Yalaoui, L. Amodeo, E. Chatelet andA. Zeblah. 2010. "Efficient combined immune-decomposition algorithm for optimal buffer allocation in production lines for throughput and profit maximization." *Computers and Operations Research* 37: 611-620.

Matondang, M. Z., and M.I. Jambak. 2010. "Soft computing in optimizing assembly lines balancing."*Journal of Computer Science* 6 (2): 141-162.

Miller, B.L. and D.E. Goldberg. 1995. "Genetic algorithms, tournament selection, and the effects of noise." *Complex Systems* 9 (3): 193-212.

Nahas, N., D. Ait-Kadi, and M. Nourelfath. 2006. "A new approach for buffer allocation in unreliable production lines." *International Journal of Production Economics* 103: 873-881.

Papadopoulos, C.T., J. Li, and Michael E. J. O'Kelly. 2019. "A Classification and Review of Timed Markov Models of Manufacturing Systems." *Computers and Industrial Engineering* 128: 219-244.

Pavai, G. and T. V. Geetha. 2019. "New crossover operators using dominance and co-dominance principles for faster convergence of genetic algorithms." *Soft Computing*23 (11): 3661-3686.

Robbins, H., and S. Monro. 1951. "A stochastic approximation method." *Annals of Mathematical Statistics* 22: 400-407.

Schindl, D., and N. Zufferey. 2015. "A learning tabu search for a truck allocation problem with linear and nonlinear cost components" *Naval Research Logistics* 62(1): 32-45.

Shi, L., and S. Men. 2003. "Optimal buffer allocation in production lines." IIE Transactions 35: 1-10.

Spinellis, D. D., and C.T. Papadopoulos. 2000. "Stochastic algorithms for buffer allocation in reliable production lines." *Mathematical Problems in Engineering* 5: 441-458.

Spinellis, D.D., C.T. Papadopoulos, and J.M. MacGregor. 2000. "Large production line optimization using simulated annealing." *International Journal of Production Research* 38: 509-541.

Suri, R. 1989. "Perturbation analysis: The state of the art and research issues explained via the GI/G/1 queue." Proc. IEEE, 77(1):114–137.

Thevenin, S., N. Zufferey, and J.-Y. Potvin. 2017. "Makespan minimization for a parallel machine scheduling problem with preemption and job incompatibility." *International Journal of Production Research* 55(6), 1588-1606.

Thevenin, S., N. Zufferey, and M. Widmer. 2016. "Order acceptance and scheduling with earliness and tardiness penalties." *Journal of Heuristics* 22(6), 849-890.

Thevenin, S., and N. Zufferey. 2019. "Learning Variable Neighborhood Search for a Scheduling Problem with Time Windows and Rejections." *Discrete Applied Mathematics* 261, 344-353.

Tomassini, M. 1999. "Parallel and Distributed Evolutionary Algorithms: A Review." *Evolutionary Algorithms in Engineering and Computer Science*, 113-133.

Turki, S., S. Hennequin, and N. Sauer. 2013. "Perturbation analysis for continuous and discrete flow models: a study of the delivery time impact on the optimal buffer level." *International Journal of Production Research* 51(13): 4011-4044.

Vergara, H.A., and D.S. Kim. 2009. "A new method for the placement of buffers in serial production lines." *International Journal of Production Research* 47(16, 15): 4437–4456.

Weiss, S., J.A. Schwarz, and R. Stolletz. 2018. "The Buffer Allocation Problem in Production Lines: Formulations, Solution Methods, and Instances." *IISE Transactions* 5854 (2): 1-30.

Xi, S., Q. Chen, J.M.G. Smith, N. Mao, A. Yu, and H. Zhang. 2020. "A new method for solving buffer allocation problem in large unbalanced production lines." *International Journal of Production Research,* 58(22), 6846-6867.

Yegul, M.F., F.S. Erenay, S. Striepe, and M. Yavuz. 2017. "Improving Configuration of Complex Production Lines via Simulation-Based Optimization." *Computers and Industrial Engineering* 109: 295-312.

Zandieh, M., M.N., Joreir-Ahmadi, and A. Fadaei-Rafsanjani. 2017. "Buffer allocation problem and preventive maintenance planning in non-homogenous unreliable production lines." *International Journal of Advanced Manufacturing* 91: 2581-2593.

Zhao, C., J. Li, N. Huang, and J.A. Horst. 2017. "Flexible Serial Lines with Setups: Analysis, Improvement, and Application." *IEEE Robotics and Automation Letters* 2 (1): 120-127.

Zufferey, N. 2012. "Metaheuristics: Some Principles for an Efficient Design." *Computer Technology and Applications* 3: 446-462.

Chapter 4

Simultaneous allocation of buffer capacities and service times in unreliable production lines

4.1 Introduction

Serial production lines are widely employed for mass and batch production systems. These lines are composed of machines connected in series with buffers between each pair of adjacent machines. Items/units (products, components, or parts) move through these machines according to predefined sequences. In such production lines, material flow disruptions can be caused by different factors such as service time variations, machine repair cycles, machine failures, or specific events linked to buffer levels such as starvation or blocking. When these types of disruptions occur, the performance of the production lines is impaired, and their efficiency is reduced since even a small change in the design parameters may lead to significant performance losses. Therefore, allocating optimal buffer capacities and the assignment of appropriate service time to each machine is fundamental to enhance the production rate (PR) and to reduce investment costs (either in the form of work-in-process inventory costs, floor space use or capital investment). In addition, considering a joint Buffer and Service Time allocation Problem (BSTAP) is a challenging problem. The computational complexity of this problem is NP-Hard, as considering the Buffer Allocation Problem (BAP) is in itself NP-Hard (MacGregor and Cruz, 2005; Dolgui et al., 2013).

Several studies have been conducted on production-lines design and optimization. Much of the research has focused on the BAP. It has been a central theme that has attracted attention over the last 30 years (Papadopoulos et al., 2019). Demir et al. (2014) and Weiss et al. (2019) have reviewed in detail the literature on BAPs. Despite the substantial studies dedicated to the design of such systems, only a few publications deal with the BSTAP.

Furthermore, most of the methods that are related to the design of manufacturing systems are based on simulation techniques. They require a high number of iterations or replications and a huge amount of time, which makes their implementation in industry difficult (Shi and Men, 2003). To reduce the time required to reach convergence, this chapter uses Perturbation-

Analysis (PA) methods that can reach good solutions on the basis of a single simulation run of the model, which reduces the convergence time (Ho and Cao, 2012).

Considering the simultaneous allocation of buffer capacities and service times, a contribution of this chapter is to combine the Finite Perturbation Analysis (FPA) and the Genetic Algorithm (GA). Indeed, such methods have been proved to be efficient in the case of BAP (Kassoul et al., 2021). In line with other techniques having a learning mechanism (Schindl and Zufferey, 2015; Thevenin and Zufferey, 2019), GA has a good exploration ability as it can quickly generate a variety of solutions that are spread over a large portion of the solution space. In contrast, FPA has a good exploitation ability as it can intensify the search around some promising solutions by determining an exploration direction in the search space provided by the gradient of the PR with respect to the system's parameters. Another goal of this chapter is to develop (near-)optimal solutions for the BSTAP in unreliable production lines for various configurations and different system sizes. The impact of the topology of the (near-)optimal buffer-capacities and service-time allocations is one of the main goals of this study. Indeed, it helps the involved decision maker in designing the production line. To the best of our knowledge, the considered problem differs from all problems in previous works since the very scarce literature on the BSTAP deals only with reliable machines.

The remainder of the chapter is organized as follows. Section 2 gives a literature review on the BSTAP. Section 3 formulates the problem in terms of a mathematical programming model and presents the detailed approach for solving the simultaneous allocation problem of both buffer capacities and service times. The development of the optimization technique and related assumptions are presented in Section 4. In Section 5, various experiments are presented. Finally, Section 6 provides conclusions and future research directions.

4.2 Literature review

This section considers only simultaneous buffer and service time allocation problems, giving the fact that the literature addressing BAP and service time allocation solely is rich and will not be discussed in this chapter. Buzacott and Shanthikumar (1993) propose the first study on the simultaneous allocation of buffers and service rates. They develop an analytical method for the joint optimization problem and note that the maximum PR is generally obtained for a balanced buffer and service time allocations (i.e., a uniform-as-possible buffer and service time allocations). Hillier and So (1995) consider small tandem queueing systems with a fixed total

number of buffer spaces to maximize the throughput. They formulate the problem as a continuous-time Markov chain and show that uniform-as-possible allocations are interesting solutions, as those reported by Buzacott and Shanthikumar (1993), when the total storage space to be allocated is a multiple of the number of buffers, with less buffer spaces allocated to the end buffers rather to the centre buffers of the system. Spinellis et al. (2000) present an evaluative procedure for optimizing small and large production line configurations using simulated annealing. They propose an approach for simultaneously balancing servers, buffers, and service times allocation on large reliable production lines. For the allocation of buffers, they obtain similar results as Hillier and So (1995), i.e., there is an accumulation of buffers to the ends of the line. However, the allocation of service times does not show the uniform-as-possible allocation and follows a decreasing rate across the line. Zhang et al. (2017) propose a simulation cutting method based on the decomposition technique to solve the discrete-event optimization model of the joint buffer, service time, and server allocation problem. However, the computational time to solve the problem increases significantly when many iterations are needed to reach optimal allocations. Hillier and Hillier (2006) use a cost model that considers both cost per buffer space and revenue per unit of throughput to optimize simultaneously buffer spaces and service times for small lines (up to 4 machines). The problem is formulated as a Markov chain using both Erlang and exponential processing times. They conclude that the service times of the solutions consistently satisfy a bowl allocation (i.e., the value of service times of the first and last machine is considerably larger than the other machines) when a small number of buffer spaces is used, and that the allocation is symmetric at the interior of the line. All these works conclude that, for both storage and service times, the bowl phenomenon occurs when the number of buffer spaces is large. Cruz et al. (2012) combine a generalized expansion method with a multi-objective GA for the allocation of service times and buffers by generating solution-curves for several single-objective functions. The interest of their study is not essentially in the designs of allocation. These obtained solution curves show a compromise between total service time allocation, overall buffer allocation and throughput. Ng et al. (2017) propose a multi-objective optimization approach for unpaced production lines. They analyse four performance measures of production systems, a set of optimal patterns of workload, and buffer allocation (average buffer level, average idle times, work in process, and throughput). Their main objective is to propose a methodology for the BSTAP to real complex production lines. They find that some interesting relationships among the performance measures studied are observed when a multi-objective design framework is considered.

Spieckermann et al. (2000) propose an approach based on a GA and simulated annealing integrated with a simulation model for solving a practical buffer planning problem for a car body assembly shop. Their objective function includes buffer sizes, deviations of service times from their respective upper bounds, and the variance of service times. Optimization aims to minimize the objective function by calibrating weights for the overall buffer space used and service times of each station. The proposed approach is evaluated using a real-life case study of a car manufacturer. They conclude, in agreement with the planning engineers of the car, that the simulation-based optimization is a helpful tool to enhance the design process. Tempelmeier (2003) uses a decomposition method to determine the performance evaluation and optimal buffer allocation for a real-life car body assembly shop where both variable and deterministic processing time are considered. The author sets a desired throughput (obtained before the optimization by an initial buffer allocation) and minimizes the total buffer space. Then, he fixes the value of the total buffer found to maximize the throughput (target). Finally, he reduces the service time of each station until reaching the target throughput found earlier. Cruz et al. (2018) optimize the performance of general finite single-server queueing networks as well as studying simultaneously the minimization of service times and buffer spaces with the objective of maximizing the PR. They employ a GA for generating solutions of total buffer space and service times. To improve the throughput of the system, they redistribute the total buffer while keeping the optimal service-time allocations found by reorganizing the buffers to be allocated along the line using simulated annealing. To evaluate their methodology, they consider the automotive assembly system proposed by Spieckermann et al. (2000). The authors show that improvements in throughput are achieved using the evolutionary algorithm under various scenarios. Recently, to solve the joint BSTAP for different open queueing network topologies in reliable lines, Smith (2018) uses a sequential quadratic-programming approach and examines the allocation patterns for small merge and split topologies (2 or 3 stages with up 7 machines). The main objective is to investigate the absence or persistence of allocation schemes of the buffers and service rates. The obtained allocation patterns corroborate, in one sense, those found in (Hillier and Hillier, 2006)

The literature shows a clear gap in the case of unreliable production systems. The very few studies available for the BSTAP have only focused on reliable machines. This article offers a powerful optimization approach that significantly improves upon the state of the art. Therefore, the interest of our work is twofold. On the one hand, the development of a design technique for stochastic production systems with unreliable machines based on the simultaneous design of buffer capacities and service times will enhance the research in

production-line design. On the other hand, the development of a comprehensive optimization technique using a single-simulation run will reduce the time needed for convergence, which contributes to the development of new generation of global-local optimization techniques.

4.3 Problem formulation and solution approach

4.3.1 Problem formulation

We consider open serial production lines (also denoted as flow lines, tandem lines, transfer lines) as presented in Figure 4.1, composed of n machines $(M_1, M_2, \ldots, M_n)$ connected in series and separated by $(n-1)$ buffers $(b_1, b_2, \ldots, b_{n-1})$, where b_i denotes the buffer size located between two consecutive machines M_i and M_{i+1}. Parts enter the system from the machine M_1, then move to the first buffer b_1, then to machine M_2 and so on until they reach the last machine M_n and leave the system.

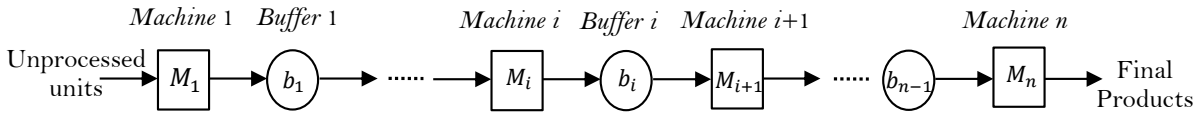


Figure 4.1. Open serial production line

The assumptions of the system are the following.

- The first machine is never starved, and the last machine is never blocked.
- Any machine is subject to breakdown but can only fail when it is up, neither starved nor blocked. The repair and failure rates of the machines are geometrically distributed.
- The transfer times of parts from machines to stocks (and inversely) are negligible.

As machines are unreliable, the state of each machine at time t can be either Down (DN) if it cannot execute any operation due to internal failure, or up (UP) if it may execute tasks/parts. We define then the failure p_i and repair r_i probabilities of the machine i as follows:

- p_i = Probability (machine i is DN for next part | i is UP for current part)
- r_i = Probability (machine i is UP for next part | i is DN for current part)

The Mean Time Between Failures (resp. the Mean Time To Repair) of M_i is $MTBF_i = t_i/p_i$ (resp. $MTTR_i = t_i/r_i$). When a machine is UP, it can be blocked or Full Output (FO) (resp. starved or Null Input (NI)) if its downstream (resp. upstream) buffer is full (resp. empty).

Consider the vector of decision variables $\theta = (\theta_1, \theta_2, \ldots, \theta_{2n-1})$ that has a dimension of $(2n-1)$, where $\{\theta_1, \theta_2, \ldots, \theta_{n-1}\} \subset \mathbb{N}$ represent the buffer capacities of available physical locations, and $\{\theta_n, \theta_{n+1}, \ldots, \theta_{2n-1}\} \subset \mathbb{R}_+$ represent the service times of machines. Moreover, B_{max} is a fixed nonnegative integer representing the total buffer space, T_{max} is the total time for the manufacturing of the product, and $f(\theta)$ is the mathematical function representing the Production Rate (PR) of the line.

The design problem addressed in this chapter is to allocate B_{max} over $(n-1)$ buffers and T_{max} over n machines. The objective is to maximize the average PR of the production line. As in Donohue and Spearman (1993) and in Suri and Leung (1987), we consider the assumption that the total time T_{max} can be distributed in some way throughout all the machines. The problem can be stated, in mathematical terms, as follows:

$$\text{Find:} \quad \theta = (\theta_1, \theta_2, \ldots, \theta_{2n-1}) \quad \text{to maximize} \quad f(\theta) \tag{4.1}$$

$$\text{Subject to:} \quad \sum_{i=1}^{n-1} \theta_i = B_{max}; \quad \theta_i \geq 0 \text{ and integer for } i = 1, 2, \ldots n-1 \tag{4.2}$$

$$\sum_{i=n}^{2n-1} \theta_i = T_{max}; \quad \theta_i \geq 0 \text{ for } i = n, \ldots, 2n-1 \tag{4.3}$$

4.3.2 Solution approach

The proposed approach is a global-search procedure using a genetic algorithm (GA), which is coupled with a Finite Perturbation Analysis (FPA) local search. The goal of combining GA and FPA is to benefit from the advantages of both methods. GA allows for approaching optimal solutions in a small computing time whereas FPA can improve the solutions with the Stochastic Algorithm (SA) using the same simulation for estimating the gradients (i.e., the simulation and optimization are conducted simultaneously). First, the GA operators are applied until a stopping condition is satisfied. Next, FPA iteratively replaces a current solution (found by GA) by a new one, until some stopping criterion is achieved. The detailed optimization approach is presented in Figure 4.2, where the initial population, $P = (c_1, c_2, \ldots, c_m)$, is composed of m different configurations (solutions), where each solution $c_i = (\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,j}, \ldots, \theta_{i,2n-1})$ is generated uniformly and randomly (see Kassoul et al. (2021) for more details on the way to generate such a population of random solutions). $\theta_{i,j}$ represents the j^{th} design parameter of the i^{th} configuration, where $\theta_{i,j}$ are integer (resp. real) numbers and represent the buffer capacities (resp. the service times) for $j = 1, 2, \ldots, n-1$ (resp. $j = n, \ldots, 2n-1$). To approach a (near-)optimal solution $c^* = (\theta_1^*, \theta_2^*, \ldots, \theta_{2n-1}^*)$, the new population $P' = (c_1', c_2', \ldots, c_m')$ is generated from the population P by applying the usual GA operators (i.e., selection, crossover, and mutation) for a fixed-number of generations. In this region, the

employed stochastic algorithm (SA) relies on the gradient descent technique (Robbins and Monro, 1951). SA takes the configurations of P' and calculates the PR gradients $\partial f(\theta)/\partial\theta_i$ according to the design decisions (buffer capacities and service times) $\theta_i (i = 1, \ldots, 2n-1)$.

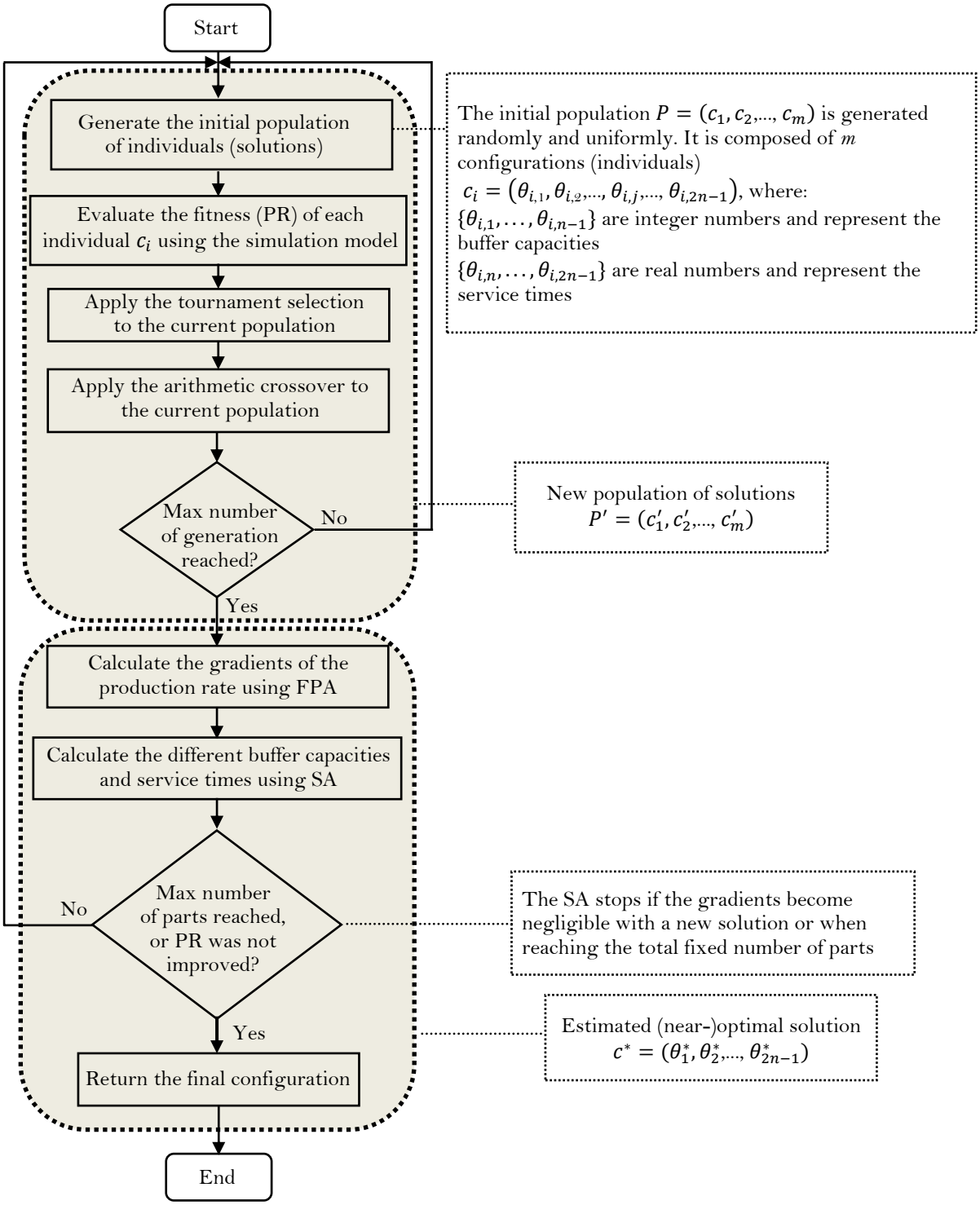


Figure 4.2. Global approach of the adopted design

Consider the original discrete-event simulation of L parts. θ_i^k represents the decision variable θ_i at the iteration k. The iteration $k+1$ allows the progression of the variables θ_i^k by simulating p parts ($p < L$) to determine a new evaluation of the gradients using FPA. The new search direction for the design solution is determined by this gradients' evaluation. For each iteration k, the projections of the gradients' vector on the following hyperplanes $\sum_{i=1}^{n-1} \theta_i = B_{max}$ and $\sum_{i=n}^{2n-1} \theta_i = T_{max}$ is estimated. In fact, for a fixed parameter θ_i, the projected gradients allow for determining the best direction for enhancing the current solution. As the search for solutions takes place in two spaces, two iterative variants of SA are constructed.

Hence: $$\theta_i^{k+1} = \theta_i^k + a_k \left(\partial f/\partial \theta_i - \frac{1}{n-1} \sum_{i=1}^{n-1} \partial f/\partial \theta_i \right) \quad (4.4)$$

and $$\theta_i^{k+1} = \theta_i^k + b_k \left(\partial f/\partial \theta_i - \frac{1}{n} \sum_{i=n}^{2n-1} \partial f/\partial \theta_i \right) \quad (4.5)$$

where

- a_k and b_k are numeric suites that verify the following conditions of convergence:

$$\lim_{k\to\infty} a_k = \lim_{k\to\infty} b_k = 0 \,, \sum_{k=1}^{\infty} a_k = \sum_{k=1}^{\infty} b_k = \infty, \ \sum_{k=1}^{\infty} {a_k}^2 < \infty \text{ and } \sum_{k=1}^{\infty} {b_k}^2 < \infty \quad (4.6)$$

- the term $\left(\partial f/\partial \theta_i - \frac{1}{n-1} \sum_{i=1}^{n-1} \partial f/\partial \theta_i \right)$ $\left(\text{resp. } \partial f/\partial \theta_i - \frac{1}{n} \sum_{i=n}^{2n-1} \partial f/\partial \theta_i \right)$ in the equation (4.4) (resp. the equation(4.5)) represents the projection of the gradient $\partial f/\partial \theta_i$ on the hyper plane constraints $\sum_{i=1}^{n-1} \theta_i = B_{max}$ (resp. $\sum_{i=n}^{2n-1} \theta_i = T_{max}$).

SA stops when there is no improvement of PR with a new solution or when the total fixed number of parts is reached.

4.4 Development of the optimization technique

4.4.1 Genetic algorithm (GA)

4.4.1.1 Background

GA is one of the commonly used population-based stochastic metaheuristics (Chaudhry and Luo, 2005). GA needs an initial population P of solutions to start with (such solutions are usually generated randomly to have a sufficient diversity of characteristics). Then, for a fixed

number of generations, GA repeats the following steps. (1) *Selection*: some solutions of P are selected according to their fitness (i.e., objective-function values). (2) *Crossover*: the selected solutions are used to generate a new population P'. (3) *Mutation*: the solutions of P' are randomly modified (individually).

4.4.1.2 Selection operator

In this study, we use a robust and effective selection mechanism commonly used by Gas, which is tournament selection (Lei et al., 2017). First, two individuals of the population are selected randomly. Next, their PR (obtained by the execution of the simulation model) are compared, and finally, the winner is selected for the next generation.

4.4.1.3 Crossover operator

The arithmetic crossover operator with a constraint criterion is used (Duman, 2018). First, two parents from the actual population, say c_1 and c_2, are selected. Then, the two parents are linearly combined to generate two new children c_1' and c_2' using equations (4.7) and (4.8), where α is a coefficient selected randomly and uniformly in the interval [0,1]. Note that an adjustment procedure is performed to fulfil constraints (4.2) (see our comments below).

$$c_1' = \alpha \cdot p_1 + (1 - \alpha) \cdot c_2 \tag{4.7}$$

$$c_2' = (1 - \alpha) \cdot p_1 + \alpha \cdot c_2 \tag{4.8}$$

4.4.1.4 Mutation operator

Since the buffer size is an integer variable, some values of the children are decreased or increased (modified) to ensure that the value of each buffer size is an integer. The adjustment procedure and the rounding mechanism for the arithmetic crossover used in this work correspond to the mutation operator (Türkyılmaz and Bulkan, 2015).

Algorithm 4.1 summarizes the steps of the proposed GA for the considered problem.

Algorithm 4.1. Genetic Algorithm

- Generate randomly and uniformly an initial population P of $(2n-1)$ individuals
- Calculate the PR of each individual using the simulation model and set $P' = \{\}$

While the new population P' does not contain $(2n-1)$ individuals, **do**

- Repeat n times the following procedure:
  Select two parents randomly
  Apply the tournament selection according to their PR values
  Clone the best-selected individual (except the last one)
- Cross two parents c_1 and c_2 using the arithmetic crossover operator
- While constraint (2) is not satisfied, apply the adjustment procedure and the rounding mechanism
- Insert the so-obtained children solutions c_1' and c_2' in the new population P'

4.4.2 Finite perturbation analysis (FPA)

We give here an overview of the generation and propagation of perturbation through FPA. A more comprehensive work on the generation and propagation rules is presented in (Cheikhrouhou, 2001). The basic idea of the Perturbation Analysis (PA) technique is to analyse a nominal sample path (from the observation of real system or single-model simulation) and to use it to construct perturbed sample-paths. The perturbed path is the result of injecting variations of decision variables into the system. Suppose that during the experiment time T, a total of L_i parts are served by machine i. Then the production rate of machine i in the nominal path is:

$$PR_i = L_i/T \tag{4.9}$$

Production lines are considered as Discrete-Event Dynamic Systems in which perturbations do not only affect the events times and durations, but also the number of parts served by the different machines, resulting in the increase or the decrease of blocking and/or starvation periods. Let Δt_i be the total event time perturbation on machine i at the end of a simulation replication. The production rate of machine i in the perturbed path is:

$$PR_i' = L_i/(T + \Delta t_i) \tag{4.10}$$

PR_i is derived from the nominal path observation, and PR_i' is calculated through the application of perturbation generation and propagation rules on the nominal path. Since perturbation generation rules are parameter specific, it is related here to the variation of buffer capacities and service times. As buffer capacities are discrete variables, a simple increase of the size could lead to a highly perturbed path with important changes in the order of the events, which does not reflect the system under consideration. Therefore, Infinitesimal Perturbation

Analysis (IPA) cannot be considered (Glasserman, 1991; Suri, 1987). We consider thus the development of FPA as an extension of PA when perturbations are finite (large amplitude) and where the system's performance is not affected by the order changes in events (Ho, 1987; Suri and Leung, 1989). Indeed, we assume that the type of future interactions that a perturbation, once introduced, can encounter and/or lead through the system in the perturbed path in and the nominal path is statistically similar (Ho, 1987). Therefore, the nominal path can be used to determine the response of the system to a perturbation. This shows the advantage provided by this technique compared to the usual techniques of gradients estimates. Let d_i be the end time of the service duration on machine i, and d_{is} be the nominal service start times of the event under consideration. Figure 4.3 shows a case where the machine M_j is NI (starved) in the nominal path, ended by a part transfer from M_i to M_j (the arrow indicates the direction of passage of the release part from one machine to another). To determine how the system may evolve in the perturbed environment, we assume that both M_i and M_j encounter the perturbation Δt_i and Δt_j (generated or propagated), respectively. The total perturbation value propagated on M_j is calculated through the formula:

$$\Delta t_j' = \text{(new starvation end time)} - \text{(old starvation end time)} \tag{4.11}$$

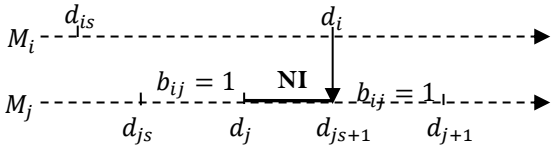


Figure 4.3. Sample path with NI case

We also define a potential starvation (resp. blocking) PNI (resp. PFO) on a server M_i as a state in which its upstream (resp. downstream) buffer contains only one (resp. $B_{max} - 1$) unit(s). When a server is starved (resp. blocked), it is no longer able to receive or to deliver, and no routing of parts can be possible (see Figures 4.4a and 4.4b, respectively).

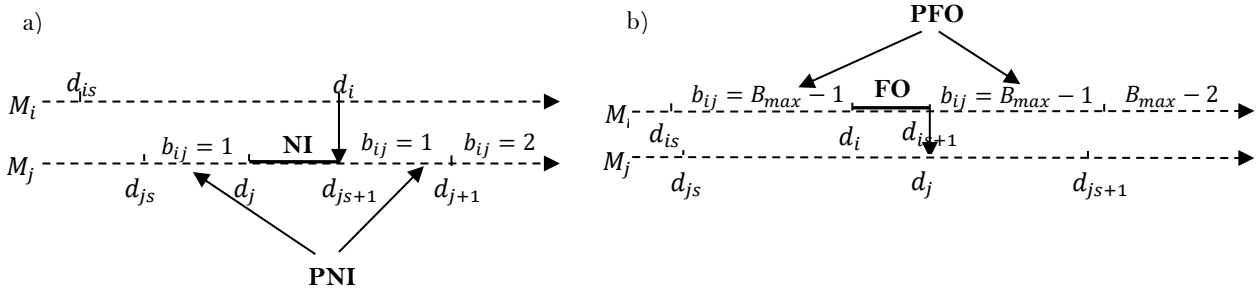


Figure 4.4. PNI and PFO periods

4.4.2.1 Case of starvation (NI)

Consider the nominal event path in Figure 4.4a. After interaction between the two sequences of perturbed events of M_i and M_j, two cases can occur.

Case 1: $d_i + \Delta t_i > d_j + \Delta t_j$. The NI period still exists in the perturbed path. The perturbation on M_i is fully propagated to M_j. The new value of the perturbation resulting on M_j is:

$$\Delta t_j' = (d_i + \Delta t_i) - d_i = \Delta t_i \tag{4.12}$$

Case 2: $d_i + \Delta t_i \leq d_j + \Delta t_j$. The NI period is eliminated and replaced by a PNI period. The perturbation is partially propagated, and the new perturbation generated is then:

$$\Delta t_j' = \left(d_j + \Delta t_j\right) - d_i = \Delta t_j + (d_j - d_i) \tag{4.13}$$

Note that the term $(d_i - d_j)$ is negative, attenuating the effect of the perturbation Δt_j on M_j, but it does not make it possible to obtain a value of the new negative perturbation $\Delta t_j'$.

4.4.2.2 Case of potential starvation (PNI)

Consider Figure 4.5, where the machine M_j is in a PNI state in the nominal path due to the unique part existing in the buffer b_{ij}.

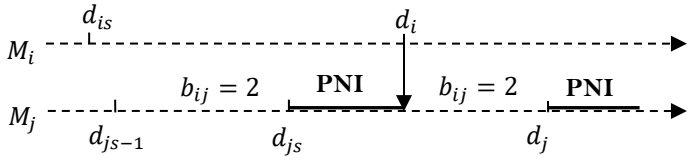


Figure 4.5. Sample path with PNI case

Interaction between servers may be transformed into real NI in the perturbed path and two cases can be distinguished.

Case 1: $d_i + \Delta t_i \leq d_j + \Delta t_j$. The PNI remains PNI, and all service event sequences retain their perturbations:

$$\Delta t_i' = \Delta t_i \tag{3.14}$$

$$\Delta t_j' = \Delta t_j \tag{4.15}$$

Case 2: $d_i + \Delta t_i > d_j + \Delta t_j$. In the perturbed path, the part transferred from M_i arrives later than the end of the service of the last part contained in the stock and thus in use by M_i. This situation creates a starvation period during which M_j remains idle. The PNI is transformed into a starvation period on M_j. The perturbation is then partially propagated, creating a NI period on M_j. Note that $(d_i - d_j)$ is negative, which asserts the phenomenon of partial propagation.

$$\Delta t'_j = (d_i + \Delta t_i) - d_j = \Delta t_i + (d_i - d_j) \tag{4.16}$$

All cases of NI or PNI periods can be summarized in the following equation, where [NI] denotes the algebraic value of the duration of a starvation interval for the machine M_j, i.e., NI $= (d_i - d_j)$.

$$\Delta t'_j = \max \{ \Delta t_i + [\text{NI}], \Delta t_j \} - \max\{0, [\text{NI}]\} \tag{4.17}$$

4.4.2.3 Case of blocking (FO)

The development of the perturbation propagation rules in the case of blocking (Figure 4.4b) is treated similarly to the case of starvation.

4.4.2.4 Case of potential blocking (PFO)

Consider Figure 4.6 where the machine M_j is in a PFO period in the nominal path.

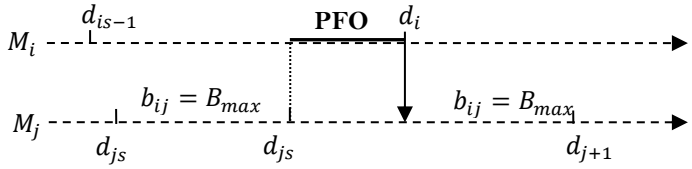


Figure 4.6. Sample path with PFO case

Two cases can be presented in the perturbed path.

Case 1: $d_i + \Delta t_i \geq d_j + \Delta t_j$. The PFO still exists in the perturbed path of M_i. There is no upstream propagation, and each machine keeps its already acquired perturbations.

$$\Delta t'_i = \Delta t_i \tag{4.18}$$

$$\Delta t'_j = \Delta t_j \tag{4.19}$$

Case 2: $d_i + \Delta t_i < d_j + \Delta t_j$. The potential FO is transformed into a blocking period on M_i. The perturbation is partially propagated from M_j, creating thus a FO period on M_i:

$$\Delta t_i' = (d_j + \Delta t_j) - d_i = \Delta t_j + (d_j - d_i) \tag{4.20}$$

All cases of FO or PFO periods can be summarized in the next equation, where [FO] is the algebraic value of the duration of a blocking interval for the machine M_i, i.e., $\mathrm{FO} = (d_i - d_j)$.

$$\Delta t_i' = \max\{\Delta t_j + [\mathrm{FO}], \Delta t_i\} - \max\{0, [\mathrm{FO}]\} \tag{4.21}$$

4.4.3 FPA algorithm

4.4.3.1 FPA algorithm for buffer capacities

Consider an observation (or a simulation) of a production line, on which a total of L parts are treated within a time duration T. We assume that all possible events during the simulation run can occur (i.e., the simulation is sufficiently long). The PR calculated over the duration T is approximated by the equation (4.22):

$$PR = L/T \tag{4.22}$$

Hence,

$$\partial PR/\partial\theta_i = \partial\left(\frac{L}{T}\right)/\partial\theta_i \tag{4.23}$$

Assuming that the stopping criterion of the simulation is the total number of parts served by the output machine of the system (i.e., L), we have

$$\frac{\partial PR}{\partial\theta_i} = \frac{\partial PR}{\partial T}\cdot\frac{\partial T}{\partial\theta_i} = -\left(\frac{L}{T^2}\right)\cdot\frac{\partial T}{\partial\theta_i} = -\left(\frac{L}{T}\right)\cdot\left(\frac{1}{T}\right)\cdot\frac{\partial \mathrm{T}}{\partial\theta_i} = -\frac{PR}{T}\cdot\frac{\partial \mathrm{T}}{\partial \theta_{\mathrm{i}}} \tag{4.24}$$

where $\partial T/\partial\theta_i$ denotes the total perturbation time that affects the system when a perturbation $\Delta\theta_i$ is introduced. In this study, the perturbation is one unit stock capacity related to each relevant event. The prediction of the impact of the perturbation concerns the event's persistence in the perturbed path and its duration. Thanks to the propagation rules of FPA, this incrementation of one unit is virtual and used only to evaluate the PR gradient and the value of $\partial T/\partial\theta_i$ as follows:

$$\frac{\partial PR}{\partial\theta_i} = -\frac{PR}{T}\cdot\frac{\partial T}{\partial\theta_i} = -\frac{PR}{T}\cdot\Delta t_n \tag{4.25}$$

where Δt_n represents the total temporal perturbation acquired by the output machine M_n after the treatment of L parts, which is also the total time gain (or loss) that is cumulated at the output machine M_n. In other ways, Δt_n is an estimator of $\frac{\partial T}{\partial \theta_i}$. Algorithm 4.2 is a simplified FPA algorithm for buffer capacities.

Algorithm 4.2. FPA for buffer capacities

Generation of perturbations

Step 1

- **For** $i = 1$ to $n - 1$, **do:** $Sum_{i+1} = 0$; Select θ_i (*initialization of accumulators* Sum_{i+1})

Step 2

- **For** « $M_i(t)$ is FO for the first time », **do:** $b_i = b_i + \Delta b_i$; $Sum_i = t_i \cdot \Delta b_i$
  (b_i *is the buffer size located between two consecutive machines* M_i *and* M_{i+1}*; in this chapter* $\Delta b_i = 1$)

Propagation of perturbation

Step 3.a

- **For** any event « $M_j(t)$ is NI » (resp. PNI), **do:** $Sum_j = max\{Sum_i + [NI], Sum_j\} - max\{0, [NI]\}$

Step 3.b (*does not apply at the first variation of the stock*)

- **For** any event « $M_i(t)$ is FO » (resp. PFO), **do:** $Sum_i = max\{Sum_j + [FO], Sum_i\} - max\{0, [FO]\}$

Step 4

If $i = n$ (last machine) **then**

- $\frac{\partial PR}{\partial b_{ij}} = -\left(\frac{PR}{T}\right) \cdot Sum_n$
- $PR^* = \frac{P}{T - Sum_n}$; Stop ($PR^*$ *denotes the estimated production rate in the perturbed trajectory*)

Else Go to Step 1

4.4.3.2 FPA algorithm for service times

To estimate $\partial PR/\partial \theta_i$ (for $i = 1, \ldots, n$), we use the perturbation analysis gradient estimator. Algorithm 4.3 is a slightly modified version of the method presented by Suri and Leung (1987). The first step is the initialization of accumulators A_{ij} $(i, j = 1, \ldots, n)$, where A_{ij} are separate accumulators needed for the gradient calculations. So, at each event (end of operation on a machine), the value of the gradient is calculated and $\frac{dt_i}{d\theta_i}$ is added to A_{ij}. $\frac{dt_i}{d\theta_i}$ denotes the sample gradient of the variable t_i, where $\frac{dt_i}{d\theta_i} = 1$ since in this work $t_i = \theta_i$ (Step 2). At the end of the simulation, an estimation of $\partial PR/\partial \theta_i$ is given by the following equation, where A_{ni}

denotes the value of accumulator at the end of the simulation.

$$\partial PR/\partial \theta_i = -(PR/T) \cdot A_{ni} \tag{4.26}$$

Algorithm 4.3. FPA for service times

Step 1: (*Initialization of accumulators* A_{ij})

- $A_{ij} = 0$ for $i, j = 1, \dots, n$ (*A_{ij} are the accumulators used to calculate the gradients*)

Step 2:

At the end of an operation on machine i with a service time t_i, **do**

- $A_{ii} = A_{ii} + \frac{dt_i}{d\theta_i}$

If a part that is leaving machine i going to machine m ends at period *NI* on machine m, **then**

- $A_{mj} = A_{ij} \quad j = 1, \dots, n$

If a part leaving machine *i* going to machine m ends a period and is *FO* on machine i, **then**

- $A_{ij} = A_{mj} \quad j = 1, \dots, n$

Step 3:

- $\partial PR/\partial \theta_i = -(PR/T) \cdot A_{ni}$

4.4.4 Stochastic Algorithm (SA)

The gradient estimates resulting from FPA are injected into a SA to compute the (near-)optimal allocation of buffer capacities and service times for the production line. The SA presented in Algorithm 4.4 is based on (Robbins and Monro, 1951), where the gradients are considered from the PR function's projection on the constraint hyperplane $\sum_i \theta_i$= constant. The SA updates intermediate variables θ_i (buffer capacities and service times) at each iteration corresponding to a perturbed path generation, following the direction of the gradient. Here, a single run optimization algorithm is used, in the sense that the update procedure is activated at each end service of p units ($p < L$). The value of p is fixed empirically. The update procedure of the configuration uses the gradients calculation's technique based on FPA (Step 3). Also, we chose a_k (respectively b_k) as numerical suites of type α/k (respectively β/k), where α (resp. β) is a random constant with a chosen initial value such that the value of the θ_i after each iteration are kept in the same order. Their values determine the step size of the production rate. A precaution is proposed in Step 4 if some updated values of θ_i^{k+1} are negative. Step 5 returns the final variable values where a discretization of buffer capacities values is necessary. The algorithm stops when a (near-)optimal solution is obtained. A trade-off between the accuracy of the estimated parameter and the simulation length is acquired by the parameter

ε, set here to 10^{-4}. The benefit of the stopping criterion used is that it considers simultaneously the variations of all variables θ_i. In other words, the algorithm stops when the variation of the variables becomes negligible within the same iteration.

Algorithm 4.4. Stochastic Algorithm

Initialization $k = 1$

Step 1

- Choose initial values of θ_i^k (for $i = 1, \dots, 2n-1$)

Step 2

- Simulate $p\ (< L)$ parts for θ_i^k and estimate $\frac{\partial PR}{\partial \theta_i}$ using PA

Step 3

- $\theta_i^{k+1} = \theta_i^k + a_k(\partial PR/\partial\theta_i - \frac{1}{n-1}\sum_{i=1}^{n-1} \partial PR/\partial\theta_i)$ (*buffer capacities*)
- $\theta_i^{k+1} = \theta_i^k + b_k(\partial PR/\partial\theta_i - \frac{1}{n}\sum_{i=n}^{2n-1} \partial PR/\partial\theta_i)$ (*service times*)

Step 4

If $\theta_i^{k+1} \leq 0$ **then**

- $\theta_p^k = Arg\ min\ \theta_i^k\ ;\ \theta_p^{k+1} = Arg\ min\ \theta_i^{k+1}$
- $a \sim U\ (0,1)\ ;\ h = \left|\frac{a \cdot \theta_p^k}{\theta_p^{k+1} - \theta_p^k}\right|$
- $\theta_i^{k+1} = \theta_i^k + h \cdot a_k \cdot \left(\frac{\partial PR}{\partial \theta_i} - \frac{1}{n-1}\sum_{i=1}^{n} \frac{\partial PR}{\partial \theta_i}\right)$ (*buffer capacities*)
- $\theta_i^{k+1} = \theta_i^k + h \cdot b_k \cdot \left(\frac{\partial PR}{\partial \theta_i} - \frac{1}{n}\sum_{i=1}^{n} \frac{\partial PR}{\partial \theta_i}\right)$ (*service times*)

Step 5

If $\left|\theta_p^{k+1} - \theta_p^k\right| \leq \varepsilon,$ **then**

- $\theta_i = Anint\left(\theta_i^k\right);\quad i = 1, \dots, n-1$ (*discretization of buffer capacities*) (*Anint: an Arena operator that takes the near integer value*)
- $\theta_i = \theta_i^k;\ i = n, \dots, 2n-1$; Stop (*service times*)

If p parts are treated, **then** Stop

Else $k = k + 1$ and Go to Step 2

4.5 Experiments

Considering serial production lines with a number n of machines ranging from 5 to 40, the aim of our experiments is not to compare the PR values with the ones obtained by other techniques from the literature (such type of works has been performed a lot), but rather to compare the obtained solution patterns with the ones from other studies, and to provide managerial insights. Indeed, with respect to the literature on the PR performance (for the

BAP), GA-FPA has already been extensively and successfully tested in Kassoul et al. (2021). For these reasons, the following experiments are conducted. In Subsection 4.5.1, to measure the benefit of combining GA with FPA, we compare the two latter methods (used independently) with GA-FPA. In Subsection 4.5.2, we measure the benefit of optimizing simultaneously buffers and service times. In Subsection 4.5.3, experiments are conducted to identify the service-time and buffer allocation patterns of the GA-FPA solutions. In Subsection 4.5.4, a sensitivity analysis is proposed with respect to having a machine with a different repair time or with a different failure time.

The algorithms are implemented in Java and the models are developed by using the simulation language Arena V14.0 (Altiok and Melamed, 2010). The experiments are run on intel Core (TM) i5 CPU @ 1.9-GHz with 8 GB of RAM. In all cases, the PR average is calculated by simulating 10,000 parts with 20 runs/replications. The total service time T_{max} is $3n$ time-units for each machine. All machines are identical and are subject to breakdown. The repair and failure times are geometrically distributed with MTTR = 10 and MTBF = 70, respectively. The algorithm stops when it reaches the maximum fixed number of 20 generations or when achieving (at a given generation) a PR better than the best PR ever reached. The computation times (presented in the last column of Tables 4.3-4.4) are not discussed in detail, but the order of magnitude is 84 minutes on average for the largest instances, which is reasonable from a practical standpoint with respect to various studies in production (e.g., Respen et al. (2016), Zufferey (2016)).

4.5.1 Comparison of GA, PA, and GA-FPA

GA-FPA is compared with FPA and GA (considered independently). Table 4.1 gives the results for instances with n in {5, 10, 15, 20, 40}. Columns 1 and 2 present n and B_{max}, respectively. Next, from columns 3 to 5, the average PR is given for GA, FPA, and GA-FPA, respectively. In terms of solution quality, GA-FPA achieves the best solutions for each instance.

Table 4.1. Results of GA, FPA and GA-FPA

Parameters		Average PR		
n	B_{max}	GA	FPA	GA-FPA
5	20	0.248063	0.246847	**0.248957**
10	45	0.233728	0.224865	**0.235617**
15	70	0.227555	0.218695	**0.228398**
20	95	0.223371	0.211147	**0.226174**
40	390	0.230927	0.220034	**0.238537**

The evolutions of the average PR with respect to the number of generations for the five instances are presented in Figure 4.7. Regarding the convergence behaviour, only a few generations are needed for GA-FPA to converge to its best solutions (it is not the case for GA and FPA). We can explain this efficient convergence of GA-FPA by the exploration ability of its GA feature to quickly identify promising regions of the solution space, and by the exploitation ability of its FPA component to intensify the search in such promising regions.

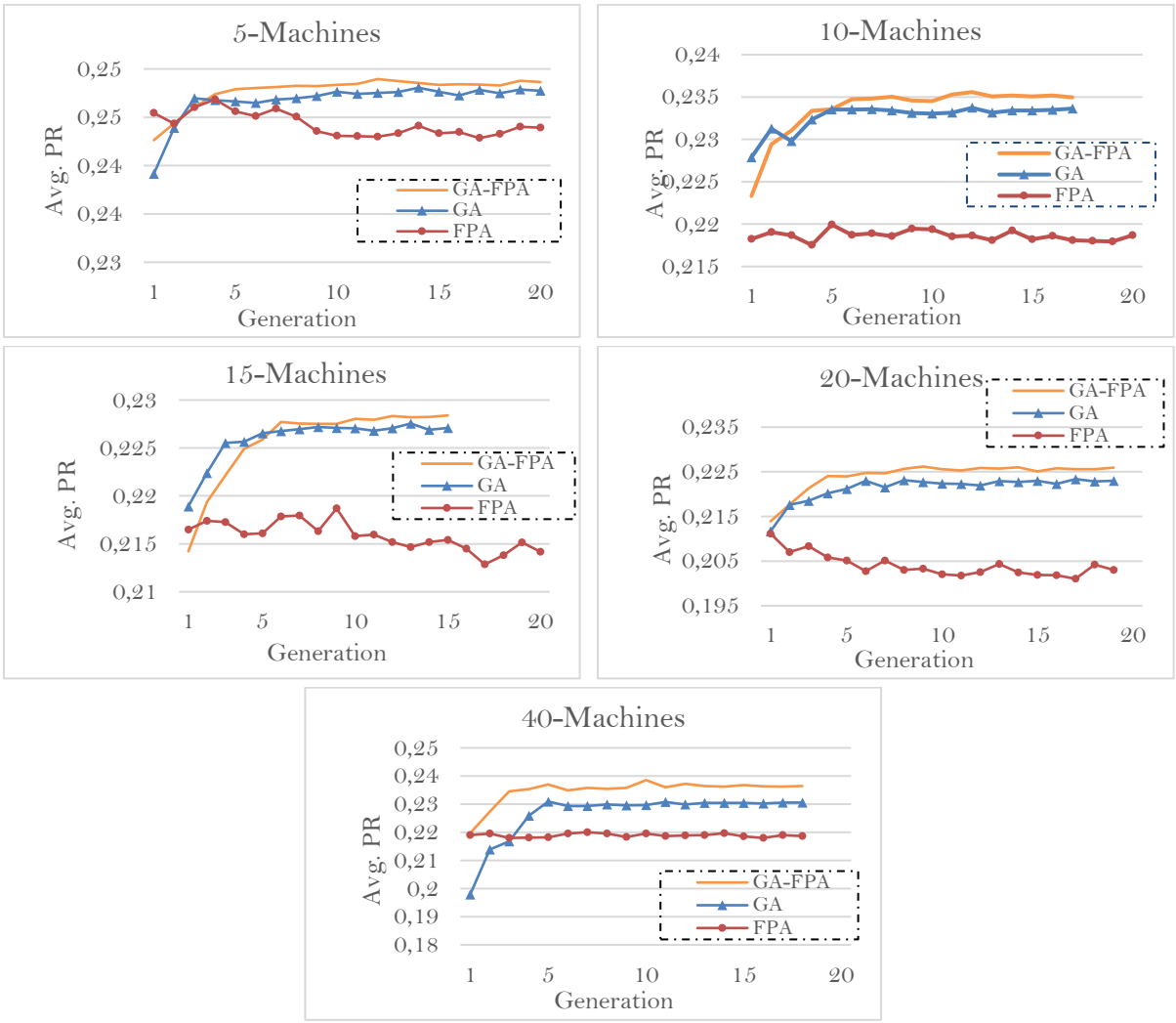


Figure 4.7. Convergence behaviour of the proposed methods

4.5.2 Benefit of optimizing simultaneously buffers and service times

To measure the benefit of the simultaneous allocation of buffers and service times, an idea is to compare the results of the simultaneous allocation with those of the buffer allocation and the service time allocation considered alone. In the case of buffer-capacity optimisation, the service times are all equal and satisfy the T_{max} constraint. Whereas in the case of service-time optimisation, the buffer capacities are as equal as possible, with some balanced adjustments to have integer numbers and to satisfy the B_{max} constraint.

Table 4.2 presents the results for instances with n in {5, 10, 15, 20, 40} machines. The first two columns present the instance parameters, followed by the average PR for service time allocation, buffer allocation, and simultaneous allocation. Note that for the instance with n = 40, only five runs (instead of 20) have been carried out.

Table 4.2. Results of service time, buffer, and simultaneous allocations

Parameters		**Average PR**		
n	B_{max}	Service time allocation	Buffer allocation	Simultaneous allocation
5	20	0.245621	0.247458	**0.248957**
10	45	0.234046	0.235018	**0.235617**
15	70	0.226720	0.227835	**0.228398**
20	95	0.225672	0.225837	**0.226174**
40	390	0.237693	0.237696	**0.238537**

We can easily observe that the joint optimization of buffers and service rates obtains the best average PR results for all instances. The quantification of this benefit may help practitioners and scientists for designing and planning production lines.

4.5.3 Solution patterns obtained by GA-FPA

Tables 4.3 and 4.4 present experiments with N in {3, 5, 7} for the small instances (Table 4.3), and with n in {10, 12, 15, 20, 40} for the medium/large instances (Table 4.4). B_{max} is given in the second column. To investigate the buffer allocation scheme with respect to a uniform allocation, B_{max} is chosen to be a multiple of the number of buffers. Column 3 (resp. 4) presents the best buffer capacities b^* (resp. service times t^*) returned by GA-FPA. Note that for Table 4.4, due to the large number of indicated values, both b^* and t^* are presented in column 3 (for n > 12). Computing times are given in the last column, in seconds.

Table 4.3. Results for small production lines (involving 3, 5, and 7 machines)

n	B_{max}	b^*	t^*	CPU(s)
3	10	5 5	3.02 2.96 3.02	977
	20	10 10	3.01 2.97 3.02	1132
	30	15 15	3.02 2.98 3.00	1191
	40	20 20	3.01 2.98 3.01	805
5	20	5 6 5 4	3.02 3.02 2.99 2.98 2.99	2506
	40	10 11 10 9	3.04 2.99 2.98 2.99 3.01	1463
	60	14 17 16 13	3.05 3.00 2.97 2.98 3.00	2018
	80	19 22 20 19	3.05 2.99 3.00 2.99 3.00	1469
7	30	3 5 7 7 6 2	3.03 3.02 2.96 3.00 2.98 3.00 3.03	2828
	60	9 10 11 11 10 9	3.02 2.99 2.98 3.00 3.00 2.99 3.02	2599
	90	15 15 16 16 14 14	3.04 2.98 3.00 2.98 3.00 2.97 3.03	1874
	120	20 20 20 21 20 19	3.02 2.97 3.00 2.99 3.01 2.99 3.02	2696

Table 4.4. Results for medium/large production lines (involving 12, 15, 20, and 40 machines)

n	B_{max}	b^*	t^*	CPU (s)
10	45	4 5 5 5 5 6 5 5 5	3.02 3.01 2.97 3.01 3.00 2.98 3.00 2.98 3.02 3.01	2183
	90	10 11 9 9 10 11 11 9 10	3.02 2.97 3.03 2.99 3.00 3.00 2.99 2.97 3.03 3.00	2249
	135	15 17 15 15 14 16 16 14 15	3.03 2.99 2.98 3.01 3.00 2.99 3.00 2.99 3.00 3.02	2353
	180	19 21 18 22 21 21 20 19 19	3.04 2.99 2.97 3.00 2.98 3.01 2.99 2.99 2.98 3.04	2269
n	B_{max}	b^*	t^*	
12	55	4 4 6 6 4 5 5 6 6 5 4	3.01 2.99 2.98 2.96 3.00 3.01 3.02 2.97 3.00 3.01 3.01 3.04	2219
	110	10 10 9 10 10 10 11 10 10 11 9	3.02 2.99 3.00 3.01 2.97 2.99 3.01 2.99 3.01 2.97 3.01 3.02	2219
	165	15 15 16 16 16 16 14 13 14 15 15	3.00 2.99 3.03 2.99 2.98 3.02 2.99 2.98 3.00 3.00 3.01 3.01	2591
	220	20 20 21 19 21 19 21 21 19 20 19	3.00 3.02 2.99 3.01 3.02 2.99 2.96 3.00 3.00 2.99 3.00 3.02	3378
n	B_{max}	b^*		
15	70	4 4 5 5 6 5 5 5 6 6 5 5 5 4		2806
	140	10 10 8 10 12 10 10 11 11 10 9 10 10 9		3118
	210	14 15 14 15 16 16 16 15 16 15 14 15 15 14		2647
	280	20 20 18 22 20 22 21 20 19 20 19 20 20 19		2426
	B_{max}	t^*		
	70	3.00 3.03 2.97 3.03 2.98 2.99 3.01 2.96 3.01 3.02 3.01 2.99 3.00 3.00 3.00		
	140	3.00 3.01 3.03 3.01 3.01 3.00 3.01 2.98 3.02 3.02 2.98 2.98 2.97 2.99 3.00		
	210	2.99 3.03 3.01 3.01 3.01 2.98 2.98 3.01 2.98 3.02 2.99 2.98 2.98 3.02 3.01		
	280	3.03 2.99 3.00 2.99 3.01 3.00 2.98 3.02 3.01 2.97 3.01 2.98 3.00 3.00 3.01		
n	B_{max}	b^*		
20	95	4 5 5 6 5 4 5 7 5 4 5 6 6 4 4 5 7 5 3		4921
	190	10 9 10 9 10 10 10 9 9 12 10 11 10 10 9 11 12 9 10		5061
	285	15 16 14 16 13 15 16 16 17 15 15 15 14 16 14 14 14 15 15		2928
	380	20 22 21 20 21 21 20 20 19 20 20 20 21 19 19 19 19 20 19		3558
	B_{max}	t^*		
	95	3.02 2.97 2.97 3.01 3.02 3.00 3.04 3.00 3.02 2.98 3.00 2.99 2.96 3.03 3.02 3.00 2.95 2.99 3.01 3.02		
	190	3.02 3.01 3.02 3.00 2.98 2.98 3.00 2.98 3.00 3.00 3.00 3.01 2.96 3.02 2.97 2.99 3.05 3.02 2.98 3.01		
	285	3.01 3.01 2.99 3.02 3.03 3.02 3.04 2.97 3.00 2.98 2.97 3.00 2.99 2.97 3.01 3.02 3.01 2.99 2.98 2.99		
	380	3.02 2.99 3.01 2.98 3.03 3.02 2.97 2.99 3.04 3.04 2.98 2.99 2.99 2.99 2.99 3.03 2.95 3.02 2.98 2.99		
n	B_{max}	b^*		
40	195	5 5 6 6 5 6 4 4 6 6 4 6 4 6 5 5 5 5 5 6		5759
		5 5 6 4 5 5 6 4 4 5 5 4 4 6 5 5 4 5 4		
	390	10 10 9 10 9 10 11 10 10 10 10 11 11 9 11 9 10 10 10 9		5101
		9 10 10 9 9 11 12 10 10 10 11 9 9 9 11 10 11 12 9		
	585	15 16 16 14 15 14 16 14 15 16 15 15 15 14 14 17 15 15 16 15		4179
		15 15 14 17 14 15 14 14 15 17 15 14 15 14 15 16 14 15 15		

780		19	21	18	20	20	19	19	22	20	19	22	21	20	23	19	19	19	21	20	21	5166
		20	20	20	20	21	19	20	18	20	20	21	20	20	20	20	20	20	20	19		

B_{max}	t^*																			
195	2.98	3.02	2.98	3.02	3.01	3.02	2.98	3.03	3.02	2.98	3.01	3.00	2.98	2.98	3.00	3.01	3.00	3.00	3.02	2.99
	3.01	3.02	2.98	3.03	3.02	2.99	2.97	2.99	3.02	3.00	3.00	3.00	2.98	2.97	3.00	3.01	2.99	3.01	2.99	2.99
390	3.03	2.98	2.97	2.98	2.97	2.98	3.02	3.02	2.99	3.00	3.02	2.96	3.01	3.01	2.98	2.98	3.01	2.99	3.01	2.98
	2.99	3.01	3.05	3.00	3.02	3.03	3.06	2.98	3.03	3.00	3.00	2.99	2.97	2.99	2.99	2.98	3.01	2.98	3.01	3.02
585	2.98	2.97	3.00	2.99	3.00	2.97	3.00	3.02	2.97	3.01	3.00	3.00	2.98	3.00	2.97	2.98	2.99	3.01	3.02	2.98
	3.00	3.03	3.03	3.03	3.02	2.96	2.97	2.96	3.00	3.00	3.03	2.98	3.01	3.01	3.01	3.02	3.05	3.00	3.03	3.02
780	3.02	2.97	2.98	3.00	3.03	2.96	3.01	3.00	3.00	3.00	2.96	2.99	3.00	2.99	3.00	3.03	3.00	3.01	2.99	2.96
	3.03	3.01	3.01	3.03	3.00	2.96	2.99	3.01	3.02	3.03	3.00	2.98	2.98	2.98	3.03	3.01	2.99	3.03	2.99	3.02

Figure 4.8 provides the allocation patterns for buffer capacities throughout the available buffer spaces between machines. Eight cases are presented (three cases of small-size production lines with n in {3, 5, 7}, five cases of medium/large-size production lines with n in {10, 12, 15, 20, 40}). For each case, four values of B_{max} are tested. Likewise, Figure 4.9 presents the allocation patterns for the service times of the machines. Note that since we determine the capacity of different buffers (Figure 4.8) and the service time of different machines (Figure 4.9), we could have represented these allocations as clouds of points. However, for sake of clarity, we prefer to connect the different points and consider a representation as if it was a continuous function.

For the small instances, the results for the allocation of buffer capacities (b^* values of Tables 4.3 and 4.4, and patterns of Figure 4.8) show that there is more storage space needed in the middle of the production line and less storage space needed at the end. These pattern allocations are likely to facilitate parts flowing downstream and to avoid possible blockages. For the allocation of service times, the results (t^* values of Tables 4.3 and 4.4, and patterns of Figure 4.9) show that the machines at the end of the line need more processing time than the machines at the centre. This is certainly to facilitate the passage of parts throughout the production line and thus to avoid congestion at the centre of the line. These results corroborate the findings of Smith (2018) and Hillier and Hillier (2006). In their studies, the authors mentioned this service-time (resp. buffer-capacity) allocation pattern as bowl-shaped (resp. inverted bowl-shaped) allocation. The bowl-shape name arises from the fact that the service-time values of the first and the last machine are larger than the service times of other machines which are relatively close together.

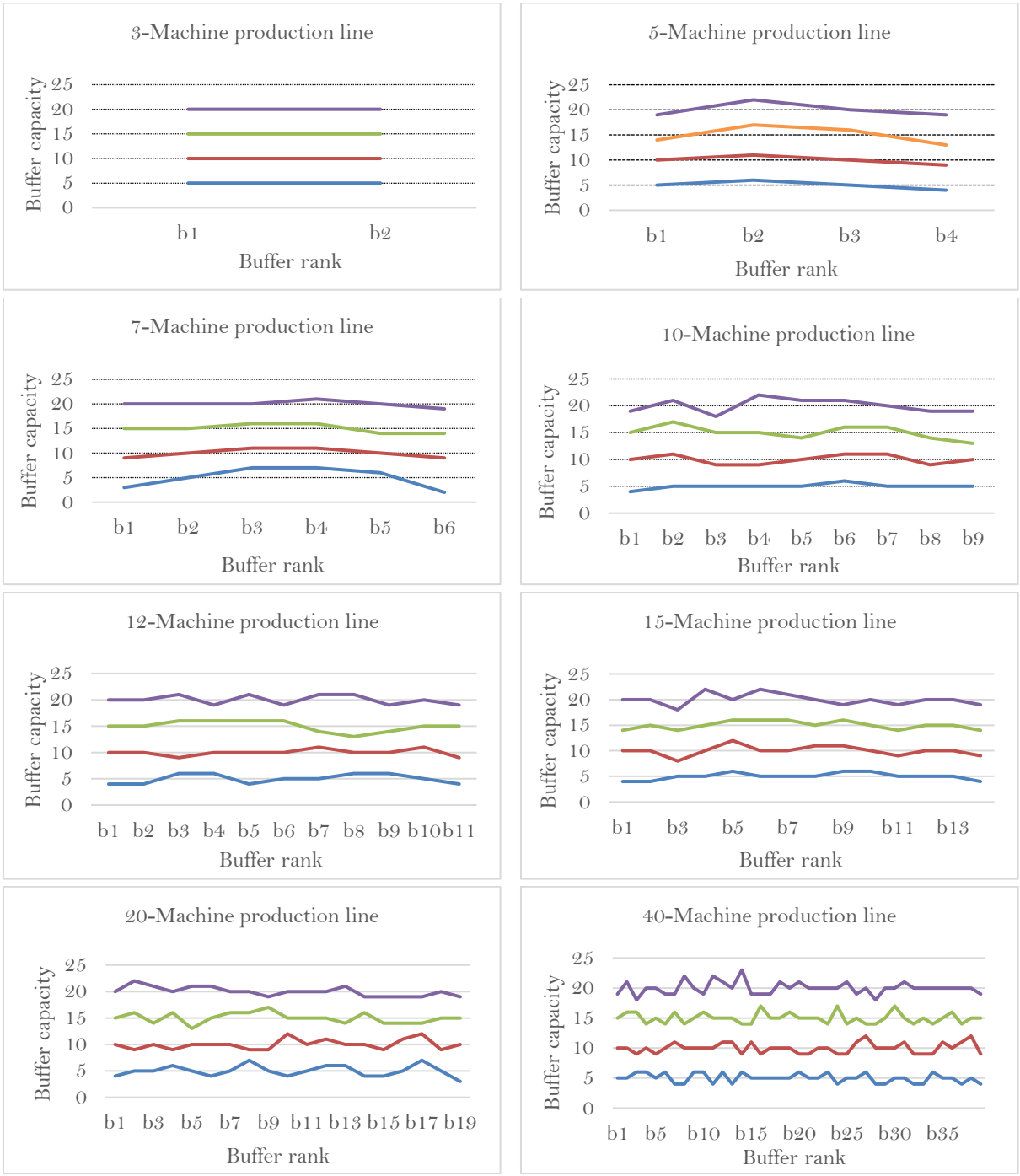


Figure 4.8. Allocation patterns for buffer capacities

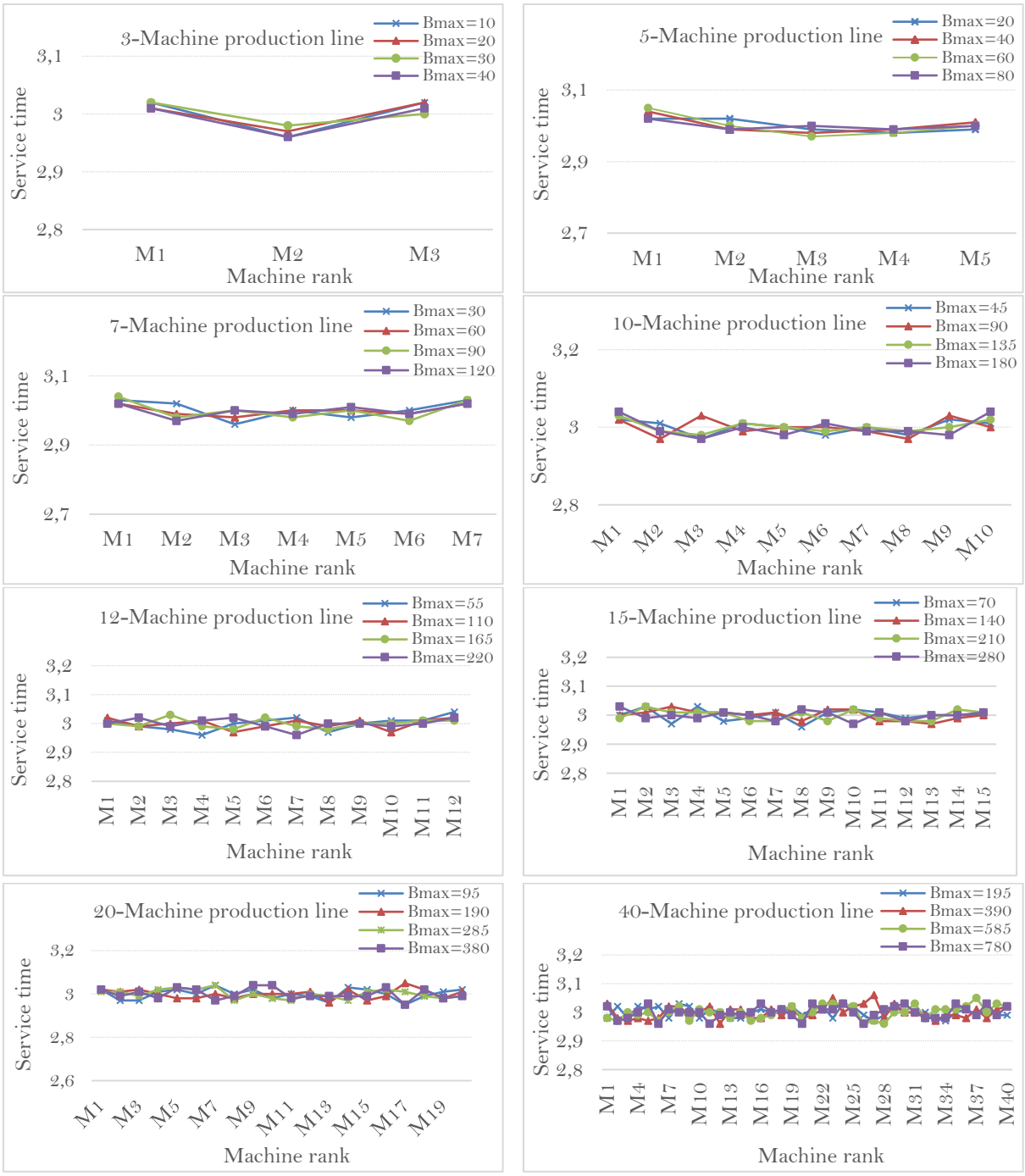


Figure 4.9. Allocation patterns for service times

For the medium/large instances (i.e., from $n = 10$), the results show that some buffers in the middle of the line require lesser storage capacities than the buffers at the ends of the production line. This buffer allocation pattern (inverted bowl-shaped) is less accentuated as the number n of machines increases. Indeed, the uniform allocation of buffers becomes more pronounced, as shown in Figure 4.7. It can also be seen that most cases favour the allocation of slightly low storage capacities at the end of the production line. This pattern allocation of buffers is likely to facilitate parts flowing downstream and thus to avoid any blockages.

Regarding the allocation of service times, the results with $n = 10$ show that the service times of the first and the last machine are slightly larger than the service times of other machines (which are still relatively close together), and some machines in the middle of the line need service times relatively like those of the end. Figure 4.8 shows that the bowl-shaped times-allocation patterns tend to disappear as n increases. Instead, a uniform allocation tends to persist. These results are similar to the findings of Spinellis et al (2000) considering the buffer allocation. However, as far as the service time is concerned, the service time allocation does not follow a uniform allocation but diminishes towards the end of the line.

In summary, the inverted bowl-shaped pattern for buffer capacities and the bowl-shaped allocation for service times seem to be diminished as the number of machines of the production line increases, and instead, both the service times and buffers allocations tend to follow roughly a uniform allocation.

4.5.4 Sensitivity analysis

Two series of experiments are presented to determine whether the allocation patterns found when all machines have the same failure and repair times are respected or, otherwise, whether there are specific buffer and service time allocation patterns when the failure and repair rates are modified. Here, the PR of the production line is calculated by treating 10,000 parts for 10 runs/replications. The results are reported in Table 4.5 and 4.6 for instances with n in {3, 10, 20}, where the total buffer size B_{max} to be assigned is presented in the second column. The fourth and fifth columns present the returned buffer capacities b^* and service times t^*, respectively.

4.5.4.1 Impact of having a machine with a different repair time

Table 4.5 presents the simultaneous buffer and service time allocations for a system where the failure times of all machines are identical and follow a geometric distribution with MTBF = 70, and where the repair times of the machines are different. For each case k, only machine k has a different repair time (MTTR = 30) compared to the other machines (MTTR = 10). The values of service time of the machine with a smaller MTTR and the corresponding downstream stock are marked in a grey font.

Table 4.5. Results if one machine has a different repair time

n	B_{max}	*Case*	b^*									t^*									
3	40	1	28				12					2.63			3.16			3.21			
		2	19				21					3.13			2.68			3.19			
		3	15				25					3.21			3.17			2.62			
n	B_{max}	*Case*	b^*									t^*									
10	180	1	21	21	21	19	18	21	19	20	20	2.48	2.94	3.05	3.08	3.08	3.08	3.10	3.08	3.05	3.06
		2	21	22	21	21	18	19	19	19	20	2.99	2.50	2.97	3.11	3.10	3.10	3.07	3.08	3.03	3.05
		3	21	20	22	19	20	20	19	20	19	3.13	2.99	2.45	2.96	3.04	3.08	3.10	3.09	3.09	3.07
		4	20	18	21	22	20	21	19	19	20	3.14	3.09	2.96	2.42	2.95	3.06	3.10	3.10	3.09	3.09
		5	20	21	19	22	20	21	20	20	17	3.18	3.15	3.08	2.95	2.42	2.98	3.02	3.06	3.09	3.07
		6	20	21	21	20	20	21	19	18	20	3.15	3.20	3.10	3.09	2.94	2.44	2.92	3.02	3.06	3.08
		7	19	20	19	21	19	21	21	21	19	3.20	3.17	3.11	3.12	3.07	2.93	2.39	2.94	3.02	3.05
		8	19	20	20	20	19	19	21	22	20	3.18	3.14	3.15	3.13	3.12	3.07	2.95	2.35	2.92	2.99
		9	19	21	20	19	20	19	21	21	20	3.17	3.19	3.18	3.12	3.11	3.06	3.07	2.90	2.35	2.85
		10	21	20	20	21	19	21	21	18	19	3.11	3.11	3.12	3.12	3.11	3.11	3.06	3.00	2.89	2.37
n	B_{max}	*Case*	b^*									t^*									
20	380	1	21	19	21	21	20	20	20	17	19	20	2.48	3.00	3.05	3.11	3.08	3.09	3.04	3.04	3.06	3.06
			19	21	20	20	20	21	20	20	21		2.66	3.02	3.07	3.04	3.02	3.03	3.03	3.09	3.03	3.00
		2	20	20	21	19	21	20	20	20	20	20	2.98	2.61	2.99	3.04	3.04	3.04	3.04	3.04	3.04	3.06
			20	22	19	20	20	19	20	20	19		3.02	2.93	3.02	3.01	3.03	3.03	3.02	3.02	2.99	3.05
		3	21	21	21	21	21	20	19	20	19	21	3.09	2.93	2.38	2.97	3.04	3.06	3.06	3.05	3.05	3.01
			19	19	19	21	20	20	19	20	19		3.03	3.00	3.04	3.06	3.07	3.04	3.06	3.04	3.01	3.01
		4	20	20	20	22	20	19	21	20	21	19	3.04	3.06	2.99	2.64	2.96	3.04	3.02	3.00	3.04	3.02
			19	19	20	20	22	21	20	18	19		3.02	3.00	3.03	3.02	3.02	2.99	3.03	3.03	3.02	3.03
		5	21	20	20	19	22	20	19	20	21	19	3.05	3.08	3.04	3.01	2.65	2.96	3.02	3.03	3.04	3.03
			21	20	19	19	20	20	20	21	19		3.02	3.02	3.02	3.02	2.97	3.02	3.01	3.01	3.01	2.99
		6	20	20	20	20	21	21	20	19	21	20	3.02	3.02	3.03	3.04	2.92	2.68	3.00	3.03	3.03	3.01
			20	19	20	20	20	19	20	19	21		3.01	3.02	3.01	3.04	3.02	3.02	3.01	3.03	3.04	3.02
		7	21	19	21	19	23	21	20	21	21	19	3.01	3.01	3.05	3.08	3.00	3.00	2.66	2.99	3.06	3.02
			19	19	18	19	20	19	21	20	20		3.02	3.04	3.02	3.01	3.01	3.02	3.02	2.98	3.00	3.00
		8	20	21	20	20	19	20	20	23	20	20	3.01	3.08	3.07	3.06	3.09	2.99	3.02	2.74	2.96	2.97
			21	19	20	20	20	18	20	19	20		3.02	3.04	3.03	3.00	3.00	2.96	2.97	3.01	2.99	2.99
		9	23	19	19	20	19	19	23	16	22	20	3.07	3.03	3.05	3.06	3.07	3.04	3.06	2.96	2.69	2.94
			21	20	20	20	20	20	21	19	19		2.98	3.02	3.03	3.03	2.99	2.99	3.05	2.97	2.99	2.98
		10	19	20	21	20	20	19	20	20	19	23	3.06	3.09	3.08	3.09	3.07	3.07	3.05	3.05	3.01	2.41
			19	21	21	20	19	21	20	19	19		2.93	3.02	3.02	2.97	3.00	2.98	3.02	3.04	3.03	3.01
		11	20	20	21	21	19	19	19	18	21	22	3.01	3.09	3.04	3.06	3.06	3.07	3.09	3.07	3.11	2.91
			21	19	20	19	20	22	19	20	20		2.40	2.93	2.95	3.02	3.04	3.05	3.04	3.00	3.04	3.02
		12	21	20	20	20	21	21	20	19	20	21	3.11	3.07	3.07	3.10	3.10	3.06	3.06	3.07	3.08	3.01
			21	23	20	18	19	20	19	18	19		2.91	2.39	2.88	3.01	3.02	3.01	3.03	3.02	2.99	3.01
		13	22	20	19	20	21	20	20	20	20	19	3.08	3.06	3.09	3.10	3.10	3.07	3.08	3.03	3.07	3.10
			19	22	22	18	19	19	19	19	22		3.02	2.88	2.26	2.92	3.00	3.03	3.03	3.00	3.03	3.05
		14	20	22	22	19	20	21	20	20	19	20	3.08	3.07	3.07	3.10	3.07	3.11	3.11	3.11	3.10	3.03
			20	19	21	22	21	20	20	18	19		3.01	3.00	2.88	2.35	2.89	2.97	3.01	3.01	3.02	3.01
		15	20	21	21	21	20	20	22	20	21	18	3.04	3.10	3.06	3.09	3.11	3.06	3.11	3.06	3.08	3.09
			19	20	18	20	20	20	19	21	19		3.07	3.11	2.97	2.83	2.34	2.81	3.02	3.04	3.01	3.00
		16	20	19	21	20	21	19	20	21	20	21	3.08	3.05	3.13	3.08	3.09	3.08	3.10	3.07	3.08	3.13
			19	18	21	20	20	21	19	21	19		3.01	3.08	3.05	3.00	2.87	2.31	2.85	2.95	3.00	2.99
		17	19	22	19	19	22	21	21	19	20	18	3.06	3.12	3.10	3.08	3.09	3.11	3.06	3.08	3.07	3.10
			21	20	20	20	18	20	23	19	19		3.07	3.09	3.02	3.03	2.98	2.86	2.31	2.84	2.95	2.98
		18	19	19	18	20	21	21	20	21	21	20	3.10	3.10	3.07	3.08	3.10	3.11	3.06	3.12	3.11	3.08
			19	19	19	19	21	21	20	22	20		3.07	3.09	3.01	3.06	3.04	2.97	2.76	2.28	2.85	2.94
		19	20	20	21	20	19	20	20	21	18	19	3.09	3.08	3.07	3.08	3.09	3.09	3.07	3.07	3.08	3.11
			20	20	21	20	21	20	20	20	20		3.09	3.09	3.07	3.04	3.01	3.00	2.95	2.82	2.30	2.80
		20	19	20	20	19	19	22	21	20	19	21	3.13	3.10	3.10	3.06	3.08	3.08	3.03	3.07	3.07	3.07
			19	19	21	21	22	18	20	19	21		3.08	3.06	3.02	3.02	3.02	3.03	3.00	2.91	2.80	2.27

The results show that the machine which has a larger repair time needs less service time than the other machines and therefore, the downstream buffer size is larger (see the grey cells). These pattern allocations are more pronounced for a production line composed of 3 machines. For example, in case 1 (resp. case 3), the value of the first stock for the obtained configuration reached is 28 (resp. 15), whereas the value of the second stock is 12 (resp. 25). We notice that

when the middle machine has a larger MTTR, the values of the two stocks are roughly similar (19 and 21 when $n = 3$). We can then observe that when n increases, the stock allocation becomes more uniform but the downstream buffer size of the machine with a larger MTTR remains in most cases slightly larger than or equal to the other buffers. In all cases, the service time of the machine with a larger MTTR is significantly smaller compared to the other machines. It can also be noted that the upstream machine and the downstream machine have a reduced service time (forming a bowl-shaped configuration around these three machines). For example, in the highlighted case 6 for $n = 10$, the best allocation reached for the service times is (3.15, 3.20, 3.10, 3.09, **2.94, 2.44, 2.92**, 3.02, 3.06, 3.08) and the best allocation proposed for the buffer capacities is (20, 21, 21, 20, 20, **21**, 19, 18, 20). We note also that the upstream machine and the downstream machine have lower service times (2.94 and 292, respectively) compared to the other machines. This certainly helps to improve the fluidity of the system and to reduce the total production time, and consequently to avoid possible blockages.

4.5.4.2 Impact of having a machine with a different failure time

Table 4.6 presents the simultaneous buffer and service time allocations for a system where all the machines have the same repair times (MTTR = 10), and for each case k, only machine k has a different failure time (MTBF = 100) compared to the other machines (MTBF = 70). The service-time values of the machine with a bigger MTBF and the downstream stock are marked in grey font. The following observations can be made.

- In all cases, the service time of the machine with the bigger MTBF is larger than the other machines.
- The service times of the downstream machines following the machine with the bigger MTBF tend to decrease towards the end of the line, and as the number of machines increases, a roughly uniform allocation occurs.
- In most cases, buffers slightly tend to accumulate towards the downstream buffer of the machine with a high MTBF, and as the number of machines increases, the buffer allocation follows a uniform allocation.

Table 4.6. Results if one machine has a different failure time

n	B_{max}	*Case*	b^*										t^*									
3	40	1	20					20					3.13			2.91				2.96		
		2	22					18					2.98			3.09				2.93		
		3	20					20					2.96			2.86				3.18		
n	B_{max}	*Case*	b^*										t^*									
10	180	1	21	21	19	21	20	19	20	19	20		3.11	3.03	2.97	2.97	2.97	3.00	3.01	3.00	2.97	2.97
		2	19	20	19	24	20	20	19	18	21		3.06	3.15	3.02	2.99	2.98	2.97	2.98	2.94	2.96	2.95
		3	21	20	20	20	22	20	20	18	19		3.06	3.03	3.13	2.98	3.00	2.97	2.98	2.98	2.92	2.95
		4	19	19	19	21	20	20	20	22	20		3.03	3.02	3.03	3.09	3.01	3.00	2.96	2.95	2.96	2.95
		5	21	19	23	18	20	20	20	19	20		3.01	3.03	2.97	2.97	3.08	2.96	2.99	3.05	2.97	2.97
		6	21	21	22	21	18	19	20	19	19		3.03	3.02	2.98	2.98	3.02	3.09	2.97	2.97	2.97	2.98
		7	21	19	21	20	20	19	21	19	20		3.04	3.05	3.03	2.98	2.97	2.99	3.10	2.94	2.94	2.96
		8	20	19	20	21	21	20	20	20	19		3.06	3.01	2.99	2.95	3.00	2.94	2.96	3.12	2.99	2.98
		9	20	21	20	21	20	20	20	19	19		3.06	3.03	2.99	2.98	2.97	2.97	2.95	2.97	3.09	2.99
		10	20	20	21	20	19	21	20	19	20		3.04	3.03	2.99	2.99	2.99	2.99	2.97	2.98	2.94	3.08
n	B_{max}	*Case*	b^*										t^*									
20	380	1	20	20	19	21	20	20	20	20	21	20	3.05	3.00	3.03	3.01	2.96	2.99	3.00	2.99	2.98	3.01
			19	20	21	19	20	20	20	21	19		2.96	2.99	3.00	3.02	2.97	2.98	3.01	3.01	3.04	3.00
		2	19	20	20	20	20	21	20	20	22	19	3.06	3.06	3.03	3.03	2.99	3.05	3.00	3.03	2.99	3.00
			19	21	19	20	20	20	19	21	20		2.94	2.97	3.02	3.00	2.9	2.99	2.99	2.98	3.00	2.97
		3	20	21	21	20	21	20	20	19	21	20	3.03	3.03	3.10	3.04	3.05	3.01	3.02	3.01	3.04	3.02
			20	21	20	20	19	20	19	19	19		3.02	2.97	3.00	3.02	2.88	2.92	2.87	2.99	2.96	3.02
		4	21	20	20	21	20	21	21	19	20	19	3.03	3.02	3.04	3.08	3.02	3.01	3.03	3.02	2.99	3.01
			21	20	20	19	19	20	20	19	20		2.98	2.99	3.02	3.00	2.98	3.01	2.97	2.94	2.95	2.91
		5	19	19	21	21	18	20	20	20	19	21	3.03	3	3.04	3.03	3.05	3.03	3.04	3.01	3.02	2.96
			21	20	19	20	19	21	22	20	20		3.02	2.98	3.01	2.96	2.98	2.97	2.98	2.97	2.95	2.97
		6	20	20	19	20	21	21	19	19	20	20	3.05	3.04	3.03	3.02	3.01	3.07	2.99	3.05	3.02	2.97
			21	20	20	20	21	19	21	19	20		2.95	3.05	3.05	3.04	2.96	2.98	2.98	2.84	2.99	2.91
		7	22	19	22	21	19	19	20	22	20	19	3.07	3.08	3.02	3.05	3.02	3.09	3.09	3.05	3.04	3.00
			19	20	20	20	21	20	19	19	19		3.02	2.98	3.01	2.97	3.01	2.99	2.88	2.78	2.94	2.91
		8	19	19	19	19	21	21	20	21	21	20	3.04	3.06	3.04	3.02	2.98	3.07	3.06	3.07	3.04	2.99
			21	23	19	19	20	22	18	19	19		2.97	2.96	2.93	3.03	2.89	2.97	3.00	2.98	2.94	2.96
		9	21	22	20	19	18	21	19	20	19	21	3.01	2.96	3.03	3.03	3.01	2.98	3.01	3.00	3.05	3.01
			20	19	21	21	20	19	19	22	19		3.00	2.99	3.03	2.99	2.97	3.03	2.98	2.97	2.94	3.01
		10	21	19	20	22	19	20	19	19	18	19	3.06	3.05	3.05	3.06	3.03	3.02	3.02	3.03	3.03	3.07
			20	23	21	21	18	20	20	20	21		3.03	2.91	3.01	2.95	2.99	2.96	2.96	2.91	2.96	2.9
		11	21	20	20	19	19	19	22	19	20	21	3.05	3.03	3.06	3.05	3.04	2.99	2.98	2.99	3.02	3.00
			21	19	21	20	19	21	20	19	20		3.07	3.03	3.00	2.99	2.96	2.96	2.94	2.96	2.94	2.94
		12	20	20	19	20	19	20	21	20	20	22	3.01	3.03	3.03	3.02	3.03	3.04	3.02	3.04	3.03	3.01
			20	20	20	20	21	21	19	19	19		3.01	3.06	3.05	3.00	3.04	2.99	2.96	2.91	2.94	2.78
		13	21	20	21	21	19	20	19	20	20	19	3.04	3.01	3.02	3.06	2.99	3.04	3.04	3.02	3.02	2.96
			20	21	20	20	19	20	19	20	21		3.00	3.05	3.09	2.92	3.03	3.03	2.95	2.86	2.98	2.89
		14	20	19	21	21	20	21	19	19	20	20	3.05	3.03	2.96	3.04	2.92	3.04	2.99	2.99	3.00	2.97
			20	19	20	20	21	20	20	19	21		2.99	2.97	2.99	3.06	3.05	3.01	2.95	2.99	2.99	3.01
		15	21	20	21	21	21	21	20	21	19	19	2.99	2.98	3.00	2.99	3.01	2.98	3.02	2.97	3.01	2.99
			18	20	19	18	20	20	20	19	22		3.01	3.00	2.94	2.97	3.04	3.03	2.97	3.04	3.03	3.03
		16	20	20	18	21	18	22	20	22	21	20	3.03	2.98	3.01	3.02	3.00	2.99	2.96	3.02	3.00	2.99
			20	20	21	20	18	20	19	20	20		3.00	3.00	3.04	2.99	3.00	3.04	3.02	2.97	2.99	2.95
		17	19	19	21	20	20	21	20	21	19	19	3.00	3.03	3.00	2.97	3.01	2.95	3.00	3.02	2.96	3.02
			21	19	21	20	19	20	21	20	20		2.97	3.00	3.02	3.02	2.97	3.00	3.04	3.01	3.01	3.00
		18	20	20	19	18	20	20	21	21	19	19	3.03	3.02	2.98	2.99	3.03	3.01	3.01	2.97	3.00	3.01
			19	21	21	20	20	19	21	20	22		3.02	3.01	2.98	2.96	3.01	3.02	2.94	3.03	3.02	2.96
		19	22	19	20	21	19	18	21	19	23	22	3.00	3.04	2.96	2.99	3.02	3.00	3.07	3.07	3.05	3.00
			19	18	21	22	19	19	17	21	20		3.00	2.99	2.99	2.99	2.95	2.91	2.99	2.98	3.09	2.91
		20	22	22	19	21	20	20	22	21	19	19	2.99	3.02	3.03	3.04	3.01	2.96	2.98	2.97	2.94	2.91
			20	21	19	19	20	18	18	21	19		2.98	2.99	3.02	2.98	3.05	2.96	3.06	3.01	3.03	3.07

4.6 Conclusion

This study presents a robust hybrid technique coupling Finite Perturbation Analysis (FPA) with Genetic Algorithm (GA) for allocating simultaneously service times and buffer capacities in production lines with unreliable machines. The processing (service) time of each machine and the capacity of each buffer (between machines) must be determined while satisfying the

total buffer-space and service-time constraints. The goal consists in maximizing the production rate of the production line. FPA (based on the estimation of the production rate's gradients) forms the core of the proposed approach, whereas GA provides its input solutions. Thanks to the exploration and diversification ability of GA, the solution-space regions of the (near-)optimal solutions can be quickly reached. Next, benefiting from the exploitation and intensification ability of FPA, these solution-space regions are deeply investigated to find better solutions. Another benefit of the proposed method relies in being able to use a single simulation for the optimization. Moreover, our experiments for instances with up to 40 machines show that the buffer and service-time allocation patterns (over the machines) corroborate previous patterns found in the literature but also exhibit some striking differences. These allocation patterns, while context dependent, are one of the most important insights for decision makers in designing production lines.

New solutions can be investigated (through the emergence of Industry 4.0 and the progress in manufacturing technologies) to improve the performance and the production rates of these systems. Indeed, the rise of intelligent and interconnected industrial robots in charge of executing system operations is helpful for monitoring inventory levels as well as tracking these different operations (including items/units transfer between operators/robots and the time needed to achieve it). In fact, the proposed approach, which allows for the simultaneous allocation of operator/robot service times and buffer capacities, offers new and interesting perspectives to enhance the performance of such systems. Another possible orientation for this research is to extend the study to other systems such as transportation systems, assembly/disassembly systems or transfer production lines.

4.7 References

Altiok, T. and B. Melamed. 2010. "Simulation modeling and analysis with Arena". *Elsevier*.

Buzacott, J. A., and Shanthikumar, J. G. (1993). Stochastic Models of Manufacturing Systems Prentice-Hall. *New Jersey*.

Chaudhry, S. S., and Luo, W. (2005). Application of genetic algorithms in production and operations management: A review. *International Journal of Production Research*, *43*(19), 4083–4101.

Cheikhrouhou, N. (2001). *Apport de l'analyse de perturbation pour le dimensionnement et le pilotage des systèmes de production* [PhD Thesis]. Grenoble INPG.

Cruz, F. R. B. da, Kendall, G., While, L., Duarte, A. R., and Brito, N. L. C. (2012). Throughput maximization of queueing networks with simultaneous minimization of service rates and buffers. *Mathematical Problems in Engineering*, *2012*.

Cruz, F. R., Duarte, A. R., and Souza, G. L. (2018). Multi-objective performance improvements of general finite single-server queueing networks. *Journal of Heuristics*, *24*(5), 757–781.

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

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

Donohue, K. L., and Spearman, M. L. (1993). Improving the design of stochastic production lines: An approach using perturbation analysis. *International Journal of Production Research*, *31*(12), 2789–2806.

Duman, S. (2018). A modified moth swarm algorithm based on an arithmetic crossover for constrained optimization and optimal power flow problems. *IEEE Access*, *6*, 45394–45416.

Glasserman, P. (1991). Structural conditions for perturbation analysis of queueing systems. *Journal of the ACM (JACM)*, *38*(4), 1005–1025.

Hillier, F. S., and So, K. C. (1995). On the optimal design of tandem queueing systems with finite buffers. *Queueing Systems*, *21*(3), 245–266.

Hillier, M. S., and Hillier, F. S. (2006). Simultaneous optimization of work and buffer space in unpaced production lines with random processing times. *IIE Transactions*, *38*(1), 39–51.

Ho, Y.-C. (1987). Performance evaluation and perturbation analysis of discrete event dynamic systems. *IEEE Transactions on Automatic Control*, *32*(7), 563–572.

Ho, Y.-C. L., and Cao, X.-R. (2012). *Perturbation analysis of discrete event dynamic systems* (Vol. 145). Springer Science and Business Media.

Kassoul, K., Cheikhrouhou, N., and Zufferey, N. (2022). Buffer allocation design for unreliable production lines using genetic algorithm and finite perturbation analysis. *International Journal of Production Research*, *60*(10), 3001–3017.

Lei, D., Zheng, Y., and Guo, X. (2017). A shuffled frog-leaping algorithm for flexible job shop scheduling with the consideration of energy consumption. *International Journal of Production Research*, *55*(11), 3126–3140.

MacGregor Smith, J., and Cruz, F. R. (2005). The buffer allocation problem for general finite buffer queueing networks. *IIE Transactions*, *37*(4), 343–365.

Ng, A. H., Shaaban, S., and Bernedixen, J. (2017). Studying unbalanced workload and buffer allocation of production systems using multi-objective optimisation. *International Journal of Production Research*, *55*(24), 7435–7451.

Papadopoulos, C. T., Li, J., and O'Kelly, M. E. (2019). A classification and review of timed Markov models of manufacturing systems. *Computers and Industrial Engineering*, *128*, 219–244.

Respen, J. Zufferey, N, and Amaldi, E. (2016). Metaheuristics for a Job Scheduling Problem with Smoothing Costs Relevant for the Car Industry, *Networks*, 67(3), 246-261.

Robbins, H., and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 400–407.

Schindl, D., and Zufferey, N. (2015). A Learning Tabu Search for a Truck Allocation Problem with Linear and Nonlinear Cost Components. *Naval Research Logistics*, 61(1), 32-45.

Shi, L., and Men, S. (2003). Optimal buffer allocation in production lines. *IIE Transactions*, *35*(1), 1–10.

Smith, J. M. (2018). Simultaneous buffer and service rate allocation in open finite queueing networks. *IISE Transactions*, *50*(3), 203–216.

Spieckermann, S., Gutenschwager, K., Heinzel, H., and Voß, S. (2000). Simulation-based optimization in the automotive industry-A case study on body shop design. *Simulation-California-*, *75*(5/6), 276–286.

Spinellis, D., Papadopoulos, C., and Smith, J. M. (2000). Large production line optimization using simulated annealing. *International Journal of Production Research*, *38*(3), 509–541.

Suri, R. (1987). Infinitesimal perturbation analysis for general discrete event systems. *Journal of the ACM (JACM)*, *34*(3), 686–717.

Suri, R., and Leung, Y. T. (1987). Single run optimization of a SIMAN model for closed loop flexible assembly systems. *Proceedings of the 19th Conference on Winter Simulation*, 738–748.

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

Thevenin, S., and Zufferey, N. (2019). Learning Variable Neighborhood Search for a Scheduling Problem with Time Windows and Rejections. *Discrete Applied Mathematics*, 261, 344-353.

Türkyılmaz, A., and Bulkan, S. (2015). A hybrid algorithm for total tardiness minimisation in flexible job shop: Genetic algorithm with parallel VNS execution. *International Journal of Production Research*, *53*(6), 1832–1848.

Weiss, S., Schwarz, J. A., and Stolletz, R. (2019). The buffer allocation problem in production lines: Formulations, solution methods, and instances. *IISE Transactions*, *51*(5), 456–485.

Zhang, M., Matta, A., Alfieri, A., and Pedrielli, G. (2017). A simulation-based benders' cuts generation for the joint workstation, workload and buffer allocation problem. *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, 1067–1072.

Zufferey, N. (2016). Tabu search approaches for two car sequencing problems with smoothing constraints, Chapter in Metaheuristics for Production Systems (Eds. L. Amodeo, E-G. Talbi, F. Yalaoui), Springer.

Conclusion

Various criteria can evaluate the efficiency of an optimization method. Among them, one can cite quality (measured by the values of the obtained solutions), speed (time required to generate competitive solutions), robustness (the standard deviation of the results being an indicator for it), and flexibility (possibility to adapt the method to different problem types). Developing an efficient method according to all these criteria is a challenging task. Nevertheless, based on the literature, one can observe that methods combining efficiently diversification and intensification mechanisms are the most promising. In this thesis, three quick, efficient, and reliable algorithms (denoted as ExPSO, APSO and GA-FPA) are proposed to solve the *Buffer Allocation Problem* (BAP) for unreliable serial production lines to maximize the production rate. GA-FPA is also extended to solve the *Buffer and Service Time Allocation Problem* (BSTAP). These two production-line design problems are NP-hard and, as the machines are unreliable (e.g., breakdowns can occur), a simulation tool is required to evaluate a solution.

ExPSO is a variant of *Particle Swarm Optimization* (PSO), using a jumping strategy with exponential function, along with dynamic parameters integrated into the velocity equation. In addition, the particle population is divided into subpopulations to maintain diversity in the particle swarm. ExPSO has the following advantages: it can escape from local minima; it can efficiently explore the solution space; it can quickly converge to efficient solutions using a concept of iteration division and a new linear decreasing cognitive parameter. ExPSO was successfully tested and compared with state-of-the-art algorithms on a set of 29 well-known benchmark functions (including unimodal, multimodal, and composite functions for different dimensional sizes) and 4 constrained engineering problems. ExPSO was also adapted to solve the BAP. In this context, ExPSO was modified by introducing a new component based on logarithmic function and dynamic parameter into the particle-velocity equation. The resulting adaptive PSO (APSO) is implemented in various problem cases consisting of production lines, ranging from $n = 3$ to $n = 100$ machines. The experiments show that the good performance of APSO does not suffer from the increase of n. This is likely to indicate a high efficiency for different problem sizes.

GA-FPA combines *Genetic Algorithm* (GA) and *Finite Perturbation Analysis* (FPA). The heart of GA-FPA is a stochastic algorithm based on the gradient estimates provided by FPA through the development of two sets of rules: the perturbation generation rules, and the perturbation propagation rules. GA-FPA was successfully adapted to the BAP and the BSTAP. Thanks to the diversification and exploration ability of GA, optimal/near-optimal solution-space regions can be quickly reached, whereas thanks to the intensification and exploitation ability of FPA, such regions can be deeply investigated to find improved solutions. The experiments conducted on instances with up to 40 machines show that the buffer and service-time allocation patterns over the machines corroborate previous patterns found in the literature but also exhibit some striking differences (particularly for medium and large instances). These allocation patterns, while context dependent, are one of the most important insights for decision makers for designing production lines.

Various research directions could be derived from this work and are all mentioned at the end of each chapter. We can mention the hybridization of other methods with the above-proposed ones (e.g., local search involving sophisticated learning mechanisms); the consideration of more complex production systems (e.g., transportation systems, split systems, and assembly systems); the optimization of other objective functions (e.g., work-in-progress minimization).