applied sciences

MDPI

Article

Embedded PSO for Solving FJSP on Embedded Environment (Industry 4.0 Era)

Rim Zarrouk [1,*], Wided Ben Daoud [2], Sami Mahfoudhi [3] and Abderrazak Jemai [4]

[1] NOCCS Laboratory, National Engineering School, Sousse University, Sousse 4054, Tunisia
[2] NTS'Com Research Unit, Sfax University, Sfax 3070, Tunisia; wided.bendaoud@isitc.u-sousse.tn
[3] Department of Management Information Systems and Production Management, College of Business and Economics, Qassim University, Buraydah 52571, Saudi Arabia; s.mahfoudhi@qu.edu.sa
[4] SERCOM Labaratory, Tunisia Polytechnic School, INSAT, Carthage University, Tunis 1080, Tunisia; abderrazek.jemai@insat.ucar.tn
* Correspondence: rim.zarrouk@isgs.u-sousse.tn

Abstract: Since of the advent of Industry 4.0, embedded systems have become an indispensable component of our life. However, one of the most significant disadvantages of these gadgets is their high power consumption. It was demonstrated that making efficient use of the device's central processing unit (CPU) enhances its energy efficiency. The use of the particle swarm optimization (PSO) over an embedded environment achieves many resource problems. Difficulties of online implementation arise primarily from the unavoidable lengthy simulation time to evaluate a candidate solution. In this paper, an embedded two-level PSO (E2L-PSO) for intelligent real-time simulation is introduced. This algorithm is proposed to be executed online and adapted to embedded applications. An automatic adaptation of the asynchronous embedded two-level PSO algorithm to CPU is completed. The Flexible Job Shop Scheduling Problem (FJSP) is selected to solve, due to its importance in the Industry 4.0 era. An analysis of the run-time performance on handling E2L-PSO over an STM32F407VG-Discovery card and a Raspberry Pi B+ card is conducted. By the experimental study, such optimization decreases the CPU time consumption by 10% to 70%, according to the CPU reduction needed (soft, medium, or hard reduction).

Keywords: particle swarm optimization; embedded environments; real-time system; flexible job shop scheduling problem; STM32F407VG-Discovery; raspberry Pi B+

Citation: Zarrouk R.; Daoud W.B.; Mahfoudhi S.; Abderrazak J. Embedded PSO for Solving FJSP on Embedded Environment (Industry 4.0 Era). *Appl. Sci.* **2022**, *12*, 2829. https://doi.org/10.3390/app12062829

Academic Editor: Giancarlo Mauri

Received: 17 December 2021
Accepted: 25 February 2022
Published: 9 March 2022

1. Introduction

Industry 4.0 refers to the fourth industrial revolution that is currently taking place in manufacturing companies. We can explain the fourth revolution (Industry 4.0) as being the implementation of new technologies and techniques (sensor, cloud, analysis of big data, etc.) in order to obtain better communication between the different objects and/or resources (people, machines, products, customers, etc.) that have a link with production for the purpose of connecting in real time to all of its resources. Figure 1 presents some Industry 4.0 pillars.

Industry 4.0 refers to a new way of organizing manufacturing processes, with the goal of establishing so-called "smart factories". Capable of increased production agility and resource allocation efficiency, these "smart factories" are setting the path for a new industrial revolution. The Internet of Things (IoT) and cyber–physical systems (CPS) are its technological foundations.

The goal of this study is to improve the performance of the particle swarm optimization (PSO) in an embedded context for addressing the Flexible Job Shop Scheduling Problem (FJSP) in smart factories. FJSP is a variant of the standard job shop scheduling problem in which each operation can be performed on many machines, each of which can perform multiple operations. The scheduling problem entails assigning processes to machines and

ordering their start times while minimizing a specific objective function, such as the total machining time or the maximal completion time of all operations. A FJSP model can be used to model a wide range of real-world challenges. The FJSP is NP-hard in the strong sense [1]. Many studies have been conducted in an attempt to solve the FJSP. For example, in [2], the authors propose an integrated and enhanced method of a dispatching algorithm based on fuzzy AHP (FAHP) and TOPSIS. In [3], the authors present a dispatching algorithm to solve a real-world case of the flexible job-shop scheduling problem with transfer batches and the objective of minimizing the average tardiness of production orders.

Figure 1. Industry 4.0.

Papers in [4,5] contain extensive research studies on the application of precise and metaheuristic techniques for solving the FJSP. Many metaheuristics are used to solve this problem. For example, in [6], an algorithm based on the principles of genetic algorithm (GA) with dynamic two-dimensional chromosomes is proposed. In [7], the authors propose a GA approach that utilizes a multi-chromosome to solve the FJSP. In a semiconductor production setting, the authors in [8] analyze stochastic flexible job-shop scheduling with limited spare resources and machine-dependent setup time, using a learning-based grey wolf optimizer. In [9], the authors use a batch-oblivious approach to solve a multiobjective complex job-shop scheduling problems. Again, in [10], a multi-objective GA is used for FJSP. Despite the fact that any metaheuristics can theoretically be applied to the FJSP, PSO is substantially easier to install and control because of its fewer parameters. PSO uses a population of particles that move around in the search space, according to simple mathematical formulae expressing the particle's position and velocity to find good, if not optimal, solutions to a problem. Each particle has a subset of particles with which it can communicate; this subset is known as swarm topology, and it can be a completely linked topology, a ring topology, a Von Neumann topology, and so on. A particle's movements are guided by its best-known position in the search space, as well as by the best-known positions of its neighbors.

Particle Swarm Optimization was originally presented as an optimization technique for static environments; however, many real-world issues are dynamic, meaning that the environment and global optimal characteristics might change over time. In this paper, we fixed the research to the dynamic environments case (i.e., embedded environment or cyber–physical systems). We adapt recent techniques which successfully address several major problems of PSO and exhibit significant performance over dynamic environments. In this paper, a technique for faster real-time simulation is introduced.

The remainder of this paper is organized as follows: the work on using PSO as a metaheuristic to address problems in embedded environments is reviewed in Section 2. Section 3 fully outlines the scheduling problem and demonstrates how the FJSP can be

solved using PSO. The PSO–FJSP variation that was employed is described in Section 4. The embedded two-level PSO (E2L-PSO) algorithm is presented in Section 5. Section 6 discusses the simulation results. The paper comes to a close with Section 7.

2. Previous Works

The entire industrial sector has entered a phase of profound change with the integration of digital technologies integrated at the heart of industrial processes. A new generation of factories was born as a result of the fourth industrial revolution. This enormous technical advance, dubbed "Cyber Factory", "Digital Factory", "Integrated Industry", "Innovative Factory", or "Industry 4.0", offers an amazing field of innovation, progress, and growth. Industry 4.0, defined by the merging of the virtual world of the delocalized internet and the actual world of industrial installations, has emerged as the reference for industrial production. In the field of embedded systems, we are currently experiencing a research trend that involves a very close integration of computing systems and physical systems, particularly with a focus on control. As a result of this tendency, a new device category known as "cyber–physical systems" has emerged (CPS). CPSs are networked infrastructures that include communication, computing, control, and sensing. The goal of CPS is to create a close link between controlled physical processes and controlling digital computing systems [11]. An excellent introduction of CPS and CPS history may be found in [12].

2.1. Cyber–Physical System

A CPS could be as simple as one sensor, a corresponding algorithm, and appropriate modalities for providing input to the physical process (for example employing an actuator). This simple scenario might be represented by a smartwatch with a pulse sensor and vibrating feedback. A CPS, on the other hand, may be represented by a very sophisticated system, such as an aircraft [13], which may be utilized for applications requiring security and safety-critical characteristics, or predictability in dynamic situations. Sensor-based communication-enabled autonomous systems are the most common CPS applications. Many wireless sensor networks, for example, monitor some aspect of the environment and transmit the processed data to a central node. Smart grid [14], autonomous car systems, medical monitoring, process control systems [15], and distributed robots are all examples of CPS.

In [14], the energy consumption of smart grid WSN (wireless sensor network) is investigated by utilizing the path operator calculus centrality based on the HSA-PSO algorithm. However, centrality in determining the route is regarded as the most difficult aspect of locating an essential node in the WSN. As a result, the path operator calculus centrality (SPOCC) is utilized to solve routing centrality problems. The SPOCC uses the harmony search algorithm (HSA) to determine the main routing path, and the Particle Swarm Optimization (PSO) algorithm to estimate high centrality nodes, thereby ensuring optimal routing with low energy consumption. The use of PSO extends the life of nodes by utilizing their dynamic capability.

In [15], a distributed algorithm is designed to perform intelligent maintenance planning for identical simultaneous multi-component machines in a job-shop manufacturing environment. The CPS-IIoT (cyber–physical systems industrial internet of things) paradigm is intuitively compatible with the algorithm design. Over PSO, the created algorithm is illustrated.

2.2. Embedded Particle Swarm Optimization

One of the most significant disadvantages of CPS devices is their high power consumption. It has been proven that making optimal use of the device's memory enhances its energy efficiency. For the resolution of optimization problems in such systems, many metaheuristics are used. In this field, Particle Swarm Optimization (PSO) has long been attracting wide attention from researchers. Many embedded systems have used PSO, such as the navigation of mobile sensors [16], virtual network embedded applications [17],

multi-robot (swarm robot) applications [18,19], hardware software partitioning problem in embedded system design [20], and dynamic virtual cellular manufacturing systems [21].

Ref. [22] presents the physically embedded PSO (pePSO) algorithm, which uses two search techniques. The swarm robots first travel around the search space, taking measurements as they approach their objectives. Second, the idea of trophallactic is converted into an algorithm and used in the search. Trophallactic is the exchange of vomited partly digested food between adults and larvae in social insect colonies. No robot-to-robot communication is used in this method. Furthermore, the robots do not need to be aware of those exact locations. Because the pePSO does not employ a primary agent to control the motions and behaviors of the swarm robots, the volume of information exchange between them is reduced. The authors give a complete overview of chaotic embedded metaheuristic optimization algorithms in [23] and detail their evolution, as well as certain advances, in addition to their integration with various methodologies and applications. Ref. [19] presents a survey on multi-robot search inspired by swarm intelligence, which classifies and discusses the theoretical advantages and downsides of previous investigations. The most appealing strategies are then analyzed and contrasted by highlighting their most important characteristics. Experiments were carried out to compare five state-of-the-art algorithms (robotic Darwinian PSO (RDPSO), extended PSO (EPSO), physically embedded PSO, etc.) for cooperative exploration tasks. The simulated experimental results show the superiority of the RDPSO. In [24], the authors suggest a PSO variation that is integrated with the notion of a micro-genetic algorithm (mGA), dubbed mGA embedded PSO, to conduct feature selection for intelligent face emotion identification. To alleviate the premature convergence problem of traditional PSO, it adds a non-replaceable memory, a small-population secondary swarm, and a collaboration of local exploitation and global exploration search mechanisms. On a complex and diverse MPSoC, the authors of [21] apply a PSO variation to handle mapping and scheduling challenges. All previous studies had only used PSO for the resolution of problems on complex and heterogeneous embedded systems.

In a recent study [25], the authors used a PSO and virtual force (VF) algorithm for solving wireless sensor network problems (optimization on the basis of providing perception service and collecting location information). In [26], a chaos-embedded PSO (CEPSO) is used to find the unknown parameters of polymer electrolyte membrane fuel cells (PEMFC) in electric vehicles and unmanned aerial vehicle applications. The authors in [27] propose a hybrid artificial neural network PSO (ANN-PSO) model in the behavior prediction of channel connectors embedded. In the multimedia domain (image processing), PSO [28] is used to optimize the scaling factors and the parameter of the improved discrete fractional angular transform in watermark schemes. A chaos-embedded PSO (CEPS) [29] is introduced, which can improve the interpolation precision and reduce the calculation time of interpolation to realize the fast reconstruction of the target 3D image based on biomechanical characteristics. In the medical imaging and deep learning domain, Ref. [30] provides automation in brain tumor segmentation with an adaptive PSO with noise removal and improved image quality.

As a result, we noticed that the current heterogeneous embedded systems are critical for high-volume markets with stringent performance requirements, and metaheuristics are frequently employed to tackle various optimization challenges in these settings. However, it comes with a slew of issues that must be solved before it can be effectively used in embedded systems. Because memory and CPU difficulties are the most prevalent problems in these systems, standard strategies for solving them generally fail. This challenge was explicitly characterized in this study as follows: given an application that must be run by a circuit, the goal is to fit that program in memory in such a way that the computation time necessary for execution is as short as possible. To overcome memory and CPU use difficulties, we offer a PSO technique (embedded two-level PSO) that is flexible to the needs of embedded systems.

2.3. Discussion

Following these brief observations from the literature (presented in Table 1), we count a multitude of studies working on the PSO to solve embedded problems. In fact, we note that the related works have flaws.

- Most studies do not deal with the use of its algorithms in embedded systems.
- Even compared to the works which envisage the adaptive PSO, the authors speak about the adaptability to a dynamic problem and not to a dynamic environment.
- The related work which uses the PSO to solve an optimization problem in an embedded environment does not place importance on the behaviors of this algorithm. Instead, they used it as a tool only, without taking into account the potential losses of CPU and memory time that we can have. In fact, they do not provide a version of PSO that is adaptable to the change of an on-board system in real time.
- Despite the importance of their contributions, none of these works address the general framework with the five types of constraints which are considered here: (1) the makespan (MS), (2) the machine load constraint (workload), (3) CPU time, (4) memory, (5) and feasibility in real time.

In this paper, we work on only two criteria: the quality of the solutions (MS) and the CPU time, in an environmental constraint (embedded environments). Both of these criteria are ensured in real-time application.

Table 1. Previous embedded particle swarm optimization (PSO) works.

Reference	Publication Year	Variant	Field of Application	Objective
[16]	2004	Basic PSO	Mobile sensors	Optimize the membership functions and rules of the fuzzy logic controller
[17]	2017	Sixteen metaheuristics (GA, ACO, PSO…)	Virtual network	Evaluate the Virtual Network Embedding (VNE) process
[18]	2008	Basic PSO	Mobile robots	Use PSO embedded into a robot swarm to find a target, in the presence of obstacles
[19]	2014	Robotic Darwinian PSO (RDPSO)	Mobile robots	Multi-robot optimization search
[20]	2008	Basic PSO	Embedded system design	Hardware software partitioning resolution
[21]	2009	Basic PSO	Production planning	Solving an extended model of dynamic virtual cellular manufacturing systems
[22]	2010	Physically-embedded PSO (pePSO)	Mobile robots	Multi-robot optimization search
[24]	2016	PSO and GA	Facial expression recognition	Conduct feature selection for Intelligent face emotion identification
[25]	2021	PSO and virtual force (VF) algorithm	Wireless sensor network	Optimization on the basis of providing perception service and collecting location information
[26]	2021	Chaos-embedded PSO (CEPSO)	Electric and aerial vehicle	Used to find the unknown parameters of polymer electrolyte membrane fuel cells (PEMFC).
[27]	2019	Hybrid PSO	Artificial neural network	Optimize behavior prediction of channel connectors
[28]	2019	Basic PSO	Multimedia, image processing	Optimize the scaling factors and the parameter of the improved discrete fractional angular transform in watermark schemes.
[29]	2020	Chaos-embedded PSO (CEPS)	Multimedia, image processing	Improve the interpolation precision and reduce the calculation time of interpolation to realize the fast reconstruction of the target 3D image.
[30]	2020	Adaptive PSO	Medical imaging and deep learning	Provide automation in brain tumor segmentation

3. Flexible Job Shop Scheduling Problem

The FJSP is formally defined in this section, followed by its PSO particle representation.

3.1. Definition

A five-tuple (J, O, M, a, d) defines the FJSP, where

- $J = \{J_1, J_2, \ldots, J_n\}$ is a series of n distinct jobs with no precedence relationship between them.
- $O = \{\{O_{11}, O_{12}, \ldots\} \cup \{O_{21}, O_{22}, \ldots\} \ldots \cup \{O_{n1}, O_{n2}, \ldots\}\}$, is the series of operations, where O_{ji} represents operation i, job j;
- $M = \{M_1, M_2, \ldots, M_k\}$ is a series of machines;
- $a : O \times M \rightarrow \{0, 1\}$, $a(O_{ji}, M_l) = 1$ if O_{ji} can be processed by M_l; and
- $d : O \times M \rightarrow N$, $d(O_{ji}, M_l)$ defines the standing of O_{ji} on M_l.

There are two types of FJSP instances: total and partial [31]. If any machine processes any action, an FJSP instance is total (noted as $FJSP^T$); otherwise, it is partial (noted as $FJSP^P$):

$$FJSP = \begin{cases} FJSP^T, & if \;\; a(O_{ji}, M_l) = 1; \;\; \forall i, j, l \\ FJSP^P, & otherwise \end{cases} \tag{1}$$

An FJSP instance's flexibility degree (FD) is defined as

$$FD = \frac{\sum\limits_{\substack{j=1\ldots|J| \\ i=1\ldots|J_j| \\ l=1\ldots|M|}} a(O_{ji}, M_l)}{|M|.|O|} \tag{2}$$

where $|J_j$ denotes the number of operations performed in job j. The makespan (MS) in Equation (3), which measures the completion time of all jobs, the robustness measure (RM) in Equation (4), which measures the schedule insensitivity to disruptions, the sum of delays (SD) in Equation (5), and the work load (WL) of all machines in Equation (6) can all be set as optimization criteria for the $FJSP$:

$$MS = max\{C_1, C_2, \ldots, C_n\} \tag{3}$$

$$RM = ST - MS \tag{4}$$

$$SD = \sum_{j=1}^{n} (C_j - E_j) \tag{5}$$

$$WL = \sum_{l=1}^{k} d(O_{ji}, M_l) \mid O_{ji} \text{ is affected to } M_l \tag{6}$$

where C_j is the completion time of job j, E_j is the minimum duration of job j, and ST is all machines' rest time (also called slack time).

Various assumptions for the $FJSP$ might be made depending on the problem scenario. The following are the most commonly held assumptions in this paper: all machines are available at time zero, and all jobs are released; the machine setup time and transportation time between operations are minor; and operations are not preemptive. For demonstration, the $FJSP^T$ instance from Table 2 is utilized. This example consists of three jobs (J_1, J_2, J_3) with three, three and four operations, respectively, and four machines (M_1, M_2, M_3, M_4). The duration of the operations on the various machines is given in the last four columns of Table 2.

Table 2. Flexible job shop scheduling problem ($FJSP^T$) instance: 3 jobs, 10 operations and 4 machines.

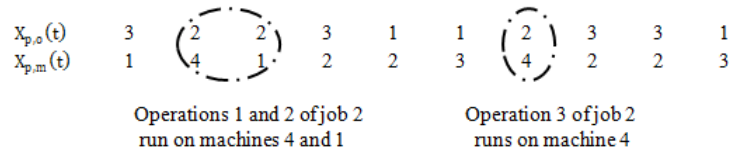
Job	Operation	Processing Time			
		M_1	M_2	M_3	M_4
1	1	1	2	3	1
1	2	1	2	3	5
1	3	3	2	2	1
2	1	3	4	1	4
2	2	3	1	3	2
2	3	2	1	3	4
3	1	4	1	2	2
3	2	1	3	3	4
3	3	2	4	1	3
3	4	1	2	1	2

3.2. Particle Representation

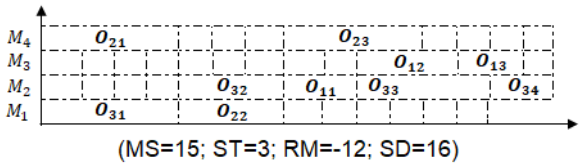
The mapping between the particle positions and the $FJSP$ solutions is referred to as particle representation. In this study, we apply the vector-form coding method as described in [32]:

A position vector of $2|O|$ integer elements in each particle p (O is the set of operations) where the initial $|O|$ items, denoted $X_{p,o}$, creates one operation scheduling order. The operation referred to by $X_{p,o}[i]$ takes precedence over the operation referred to by $X_{p,o}[i+1]$. The $X_{p,m}$ elements in the second $|O|$ element define an operations-to-machines mapping. An example of Xp,o and Xp,m is shown in Figure 2a. According to $X_{p,o}$, the first operation of J_3 is scheduled first, followed by the first and second operations of J_2, the second operation of J_3, and so on.

Figure 2a shows an example of $X_{p,o}$ and $X_{p,m}$. $X_{p,o}$ states that the first operation of J_3 is the first to be scheduled, followed by the first and second operations of J_2, then the second operation of J_3, etc. According to $X_{p,m}$, job J_1's operations are allocated to machines M_2-M_3-M_3, job J_2's operations are assigned to M_4-M_1-M_4, and job J_3's operations are assigned to M_1-M_2-M_2-M_2. The final time schedule is shown in Figure 2b.



(a) Position vector.



(b) The implicit schedule.

Figure 2. Vector form representation and the implicit schedule.

$X_{p,o}(t+1)$ and $X_{p,m}(t+1)$ are computed as follows (at time $(t+1)$): According to Equation (7), two temporary vectors are computed, $\widehat{X_{p,o}}(t+1)$ and $\widehat{X_{p,m}}(t+1)$. Then $X_{p,o}(t+1)$ is set equal to $X_{p,o}(t)$, which is sorted according to $\widehat{X_{p,o}}(t+1)$ in ascending order. The ascending order of $widehatXp,o(t+1)$ is also followed when sorting $widehatXp,m(t+1)$. $\widehat{X_{p,m}}(t+1)$ is also followed when sorting $\widehat{X_{p,o}}(t+1)$. If an element of $X_{p,m}(t+1)$ is out of range or fractional, then the closest machine number is used to replace it.

4. Used PSO-FJSP Variant

Despite the fact that any metaheuristics can theoretically be applied to the FJSP, PSO is substantially easier to install and control because of its fewer parameters. PSO uses a population of particles that move around in the search space according to simple mathematical formulae expressing the particle's position and velocity to find good, if not optimal, solutions to a problem. Each particle has a subset of particles with which it can communicate; this subset is known as swarm topology, and it can be a completely linked topology, a ring topology, a Von Neumann topology, and so on. A particle's best-known position in the search space, as well as the best-known positions of its neighbors, directs its movements. A particle p obtains a new position vector, $X_p(t+1)$, and a new velocity vector, $V_p(t+1)$, at each instant, calculated as follows:

$$V_p(t+1) = wV_p(t) + c_1R_1(Xbest_p(t) - X_p(t)) + c_2R_2(Xnbest_p(t) - X_p(t)) \quad (7)$$

$$X_p(t+1) = X_p(t) + V_p(t+1) \quad (8)$$

where w is a parameter called the "inertia weight"; c_1 and c_2 are two parameters called the "cognitive factor" (or self-recognition factor) and "social factor" (or social-component factor), respectively; R_1 and R_2 are two square diagonal matrices in which the entries on the main diagonal are random numbers in the interval [0, 1] The best location attained by a particle p up to time t is called $Xbest_p(t)$. The neighborhood best, also known as $Xnbest_p(t)$, is the best location that p's neighbors have discovered. The second level utilizes an adaptive inertia weight (Equation (9)), whereas the first level uses a time-varying inertia weight (Equation (10)):

$$w_p(t) = \begin{cases} w_{min} + \frac{(w_{max}-w_{min})(pBest(t-1)-minBest(t))}{avgBest(t)-minBest(t)}, & if\ pBest \leqslant avgBest(t) \\ w_{max} & ,otherwise \end{cases} \quad (9)$$

$$w(t) = w_{min} + (w_{max} - w_{min})\frac{t}{I_1} \quad (10)$$

$$c_1(t) = 0.5\ (w_p(t)+1)^2 \quad (11)$$

$$c_2(t) = min(4, 2(w_{min}+1)) - 0.5\ (w_p(t)+1)^2 - 0.000001 \quad (12)$$

where $avgBest(t)$ (resp. $minBest(t)$) is the average (minimum) objective value at the current iteration, $w_{max} = 0.9$ and $w_{min} = 0.4$.

The two-level PSO-FJSP attempts to shrink the search space even further and increase the exploration capability. The two-level PSO-FJSP nestedly executes a PSO twice: the top level investigates multiple operations-to-machines mappings, while the lower level investigates various schedules for a given mapping. As a result, before moving on to another mapping, numerous schedules of the same mapping are evaluated sequentially.

Particles are coded as follows: The location vector $X_{p,m}$ of an upper-level particle p of $|O|$ elements models one operations-to-machines mapping. The indices of $X_{p,m}$ and the operations of the jobs have a one-to-one prefixed correspondence. A lower-level particle has a position vector $X_{p,s}$ of $|O|$ elements, which models the order of operations mapped to each machine. Between the indices of $X_{p,s}$ and machines, there is a one-to-one prefixed correspondence.

Let S_1 and I_1 (S_2 and I_2) represent the number of particles and iterations utilized in level one (level two), respectively. The values assigned to S_1 and I_1 are determined by the degree of flexibility of an FJSP instance. The smaller values of S_1 and I_1 are the lower flexibility degree. The values allocated to S_2 and I_2 may be fixed or adaptive to the lower bound value found. If the gap between the current best objective value and the lower bound is large (small), then S_2 and I_2 are set high (low). The two-level PSO-FJSP flowchart is shown in Figure 3.

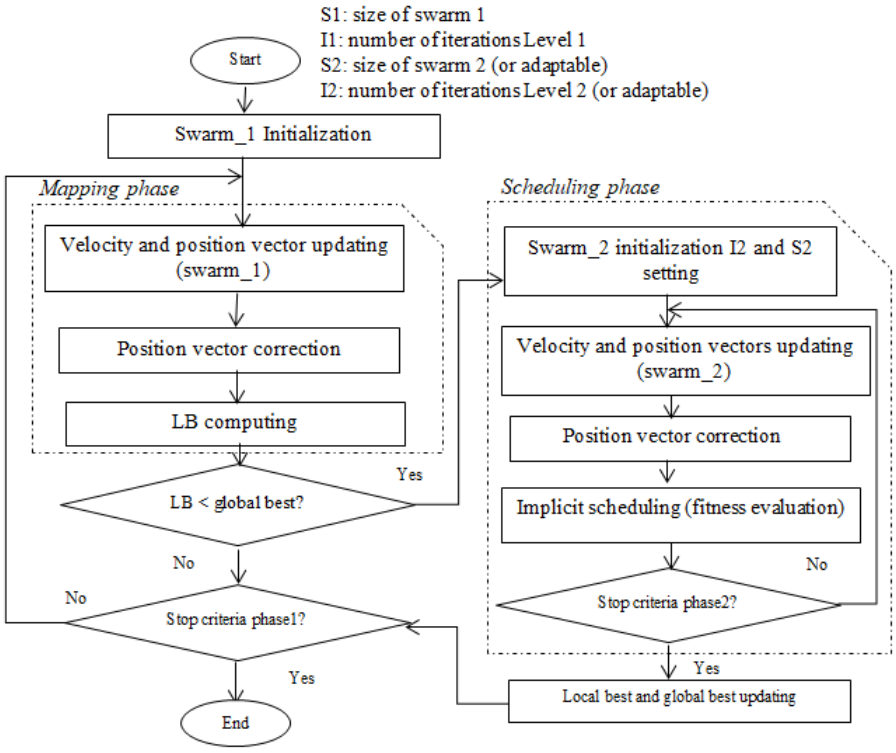


Figure 3. Flowchart of the two-level particle swarm optimizatiom—flexible job shop scheduling problem (PSO-FJSP).

5. Proposed Technique: Embedded Two-Level PSO-FJSP

Our solution is an automatic adaptation of the asynchronous two-level PSO algorithm to CPU needs by other tasks that run in parallel (embedded two-level PSO E2L-PSO).

Several variants can be envisaged, such as the suspension of certain threads or the reduction in the number of particles. The efficiency and reliability of the method always lie in the choice of particles to suspend. It is, therefore, necessary to designate the particles that return a fitness that is furthest away from the *Xnbest* (global best) because the particles, which have the best solution, are probably closer to the global optimum. In this paper, we are inspired by the migration method of dividing the swarm into subsets. We divided the swarm into a "leaders sub-swarm", a "worsts sub-swarm" and two "followers sub-swarms".

The migration process is as follows:

- "Leaders sub-swarm" began with only one particle that has the best MS in the initialization phase. "X_L" is the maximum number of particles.
- "Worsts sub-swarm" began with only one particle: the particle that has the worst MS in the initialization phase. "X_W" is the maximum number of particles.
- Particles that have the best solutions are migrated from the "followers sub-swarm" to the "leaders sub-swarm". If the number of particles in this subset is equal to "X_L" then one of these particles must randomly migrate to a "follower sub-swarm" to empty the space for the new particle.
- Particles that have the worst solutions are migrated from the "followers sub-swarm" to the "worsts sub-swarm". If the number of particles in this subset is equal to "X_W" then one of these particles must randomly migrate to a "followers sub-swarm" to empty the space for the new particle.

In this paper, we chose to work with a swarm size reducing based solution. This solution allows us to release both the CPU time and the memory. Indeed, we remove particles from the "worsts sub-swarm". Algorithm 1 presents the steps of the adaptive

E2L-PSO, where "*need*" is an external variable contains the request to free memory or the CPU need. In this approach, the minimum number of particles in the swarm is defined at the start of the program. The last particle in every phase is the latest executed particle in the last iteration. The flowchart of the E2L-PSO is given in Figure 4.

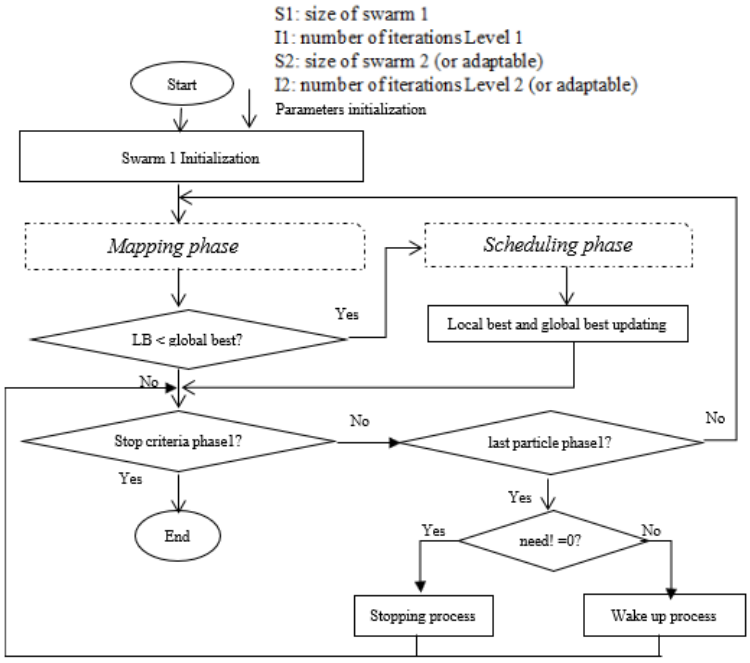


Figure 4. Flowchart of the embedded two-level PSO (E2L-PSO).

Two process are used in this algorithm: one for the stopping particle and one for waking up the particle. In the stopping process, if the number of active particles (p_{active}) is more than the number of particles to be stopped ($p_{inactive}$), and if the size of "worsts sub-swarm" is more than $p_{inactive}$, then we should randomly put $p_{inactive}$ particles in the "worsts sub-swarm" in the inactive mode. Alternatively, we should put ("worsts sub-swarm" size $-$ 1) particles in the inactive mode and a randomly put ($p_{inactive} - size_{"worsts-sub-swarm"}$) particles from the "followers sub-swarm" in the inactive mode. This process is represented in Algorithm 2.

In the wake up process, if the number of particles in the inactive mode is more than zero, so we should put 50% of $Total_{inactive}$ particles in an active mode by initializing according to the global best ($Xnbest$) coordinates and put 50% in an active mode by randomly initializing. We should add it to the "followers sub-swarm". This process is presented in Algorithm 3.

To resume, we can say that, compared to the two-level PSO presented in [32], our new proposed approach "E2L-PSO" adds four new processes:

- After swarm initialization, we divide the swarm into a sub-swarm: "leaders sub-swarm", "followers sub-swarm" and "worsts sub-swarm". So our population work under swarms.
- After each iteration, a migration method is added. Particles are migrated from the "followers sub-swarms" to the "leaders sub-swarm" and from the "followers sub-swarms" to the "worsts sub-swarm".
- A "stopping process" is added for swarm size reducing to response to the request to free memory or the CPU need (presented in Algorithm 2.
- A "wake-up process" is used if there is no need (presented in Algorithm 3).

Algorithm 1 Adaptive E2L-PSO steps

Swarm initialization
Divide the swarm into a sub-swarm: "leaders sub-swarm", "followers sub-swarm" and "worsts sub-swarm".
$Total_{inactive} = 0$.
for each iteration **do**
  for each swarm **do**
    for each particle **do**
      Mapping phase (1)
      if ($LB < Xnbest$) **then**
        Scheduling phase (2)
        $Xbest$ and $Xnbest$ update
      else
        if (!stop-criteria) **then**
          go to (1)
        end if
      end if
    end for
  end for
  Migrate particles that have best solutions from the "followers sub-swarm" to the "leaders sub-swarm"
  Migrate particles that have worst solutions from the "followers sub-swarm" to the "worsts sub-swarm".
  if ($need! = 0$) **then**
    Calculate the number of particles to be stopped ($p_{inactive}$)
    $Total_{inactive} = Total_{inactive} + p_{inactive}$.
    Stop process ($p_{inactive}$)
  else
    if the number of particles in the inactive mode ! = 0 **then**
      Wake up process ($Total_{inactive}$)
    end if
  end if
end for

Algorithm 2 Stopping process

if ($p_{active} > p_{inactive}$) **then**
  if ($p_{inactive} < size_{"worstssub-swarm"}$) **then**
    Put $p_{inactive}$ particles in the "worsts sub-swarm" randomly in inactive mode.
  else
    Put ("worsts sub-swarm" size $-$ 1) particles in inactive mode
    Randomly put ($p_{inactive} - size_{"worsts-sub-swarm"}$) particles from the "followers sub-swarm" in inactive mode.
  end if
end if

Algorithm 3 Wake up process

Put 50% of $Total_{inactive}$ particles in an active mode by initializing according to the $Xnbest$ coordinates, adding it to the "followers sub-swarm".
Put 50% of $Total_{inactive}$ particles in an active mode by randomly initializing, adding it to the "followers sub-swarm".

6. E2L-PSO—Experimental Results

Experiments were performed on micro-controller (STM32F407VGT6) and a ARM-based mono-processor nano-computer (Raspberry Pi B+):

- STM32F407VGT6 has a 32-bit ARM Cortex-M4F core, 1 MB Flash, 192 KB RAM.
- Raspberry Pi B+ has a Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor and 1 GB of RAM.

6.1. Experiment Design

The following PSO versions were designed and tested in this study:

- Standard PSO (is developed for comparison purposes only).
- The one-level PSO (is utilized for comparison purposes only).
- The two-level PSO without use of lower bound (is utilized for comparison purposes only).
- Embedded two-level PSO (E2L-PSO).

Five sets of experiments are defined:

- Experiment #1 Basic PSO vs. one-level PSO [32] vs. E2L-PSO without reduction in STM32 F407VGT6 and Raspberry Pi B+. Two comparison criteria are used: makespan as defined by Equation (3) and the used CPU time.
- Experiment #2 Adaptive E2L-PSO on Raspberry Pi B+.
  Three scenarios take place:
  - Scenario 1: "Hard reduction" 60% reduction in CPU usage.
  - Scenario 2: "Medium reduction" 40% reduction in CPU usage.
  - Scenario 3: "Soft reduction" 20% reduction in CPU usage.

  Under each scenario, we have three possible cases:
  - The reduction request has arrived after only 20% of the iterations are completed (Request_Period = 20%).
  - The reduction request has arrived after 50% of the iterations are completed (Request_Period = 50%).
  - The reduction request has arrived after 90% of the iterations are completed (Request_Period = 90%).
- Experiment #3 Adaptive E2L-PSO stability analyses. The statistical significance of the difference in performance between E2L-PSO without CPU reduction and adaptive E2L-PSO with medium reduction is determined using Mann–Whitney tests.
- Experiment #4 Adaptive E2L-PSO robustness analyses. We use the definition presented in Equation (4), where the robustness is defined as the difference of the slack time (ST) and the makespan.
- Experiment #5 Comparison with previous metaheuristics. These simulations pit the E2L-PSO against 11 different metaheuristics, 5 of which are PSO based and 6 of which are from other categories (GA, TS, etc.). All comparisons are made using the same global conditions as the prior study (iteration number, swarm size, etc.).

Each simulation is performed a total of 30 times to better measure the stability with the Mann–Whitney method.

The control parameters of E2L-PSO are summarized in Table 3.

Table 3. PSO parameters.

Hard (60% CPU)			
Request_Period	Beginning (20%)	Middle (50%)	End (90%)
c_1	1.49618 for level 1, Equation (11) for level 2	1.49618	1.49618
c_2	1.49618 for level 1, Equation (12) for level 2	1.49618	1.49618
w	Equation (10) for level 1, Equation (9) for level 2	0.6	0.6
Medium (40% CPU)			
Request_Period	Beginning (20%)	Middle (50%)	End (90%)
c_1	1.49618 for level 1, Equation (11) for level 2	1.49618 for level 1, Equation (11) for level 2	1.49618
c_2	1.49618 for level 1, Equation (12) for level 2	1.49618 for level 1, Equation (12) for level 2	1.49618
w	(10) for level 1, Equation (9) for level 2	Equation (10) for level 1, Equation (9) for level 2	0.6
Soft (20% CPU)			
Request_Period	Beginning (20%)	Middle (50%)	End (90%)
c_1	1.49618 for level 1, Equation (11) for level 2	1.49618 for level 1, Equation (11) for level 2	1.49618
c_2	1.49618 for level 1, Equation (12) for level 2	1.49618 for level 1, Equation (12) for level 2	1.49618
w	Equation (10) for level 1, Equation (9) for level 2	(10) for level 1, Equation (9) for level 2	0.6

The locations of particles are initialized as follows: using the localization strategy [31], 50 percent of the population is initialized to viable solutions (i.e., feasible schedules), and 50 percent is initialized with random values. The performance of the algorithms is assessed using 10 benchmarks [33]. The number of jobs, machines, and operations; the flexibility degree (Equation (2)); and the lower bound on makespan [34] are all listed in Table 4 for each benchmark.

Table 4. FJSP benchmarks.

Instance	#Jobs	#Machines	#Operations	FD	Lower Bound on Makespan
MK01	10	6	55	0.34	36
MK02	10	6	58	0.68	26
MK03	15	8	150	0.37	204
MK04	15	8	90	0.23	48
MK05	15	4	106	0.42	168
MK06	10	15	150	0.32	33
MK07	20	5	100	0.56	133
MK08	20	10	225	0.14	523
MK09	20	10	240	0.25	299
MK10	20	15	240	0.19	165

6.2. Simulation Results

6.2.1. Experiment #1 (Basic PSO vs. One-Level PSO vs. E2L-PSO without Reduction on STM32 F407VGT6 and Raspberry Pi B+)

The one-level PSO is presented in [32]. To better compare the different PSO variants, first, we start working with the same parameters as in [32] (I_1 = 64; I_2 = 5; S_1 = 50; S_2 = 5). In other words, the number of iterations (I, I_1, I_2) is set so that the total number of solutions visited in each version is approximately 80,000. Results are presented in Table 5 for simulations in the STM32F407VGT6 and in Table 6 for simulations in Raspberry Pi B+.

Tables 5 and 6 show that the one-level PSO produces better results for MS but not for CPU time when compared to the standard PSO. For all benchmarks, however, the E2L-PSO without reduction delivers higher performance on the two criteria. Because the size of the position vectors is reduced, the CPU time falls by 2 to 8 times.

In Tables 7 and 8, we decrease the number of visited solutions at 5000 points. We find that E2L-PSO without reduction remains the best, as it provides better results on the two criteria for all benchmarks. The CPU time decreases by 2 to 7 times.

Table 5. Comparison 1 between the three developed PSO variants on STM32 F407VGT6.

Instance	Basic PSO $I = 1454$ $S = 55$		One-Level PSO [32] $I = 1454$ $S = 55$		E2L-PSO without Reduction $\boldsymbol{I_1} = 64$ $\boldsymbol{S_2} = 50$ $\boldsymbol{I_2} = 5$ $\boldsymbol{S_2} = 5$	
	MS	CPU Time	*MS*	CPU Time	*MS*	CPU Time
4 × 5	12	68	11	30	11	21
8 × 8	20	201	16	197	14	50
10 × 10	12	433	10	313	7	52
MK01	44	378	40	815	39	101
MK02	33	410	29	738	27	140
MK03	204	3302	204	3111	204	432
MK04	70	799	62	1105	60	280
MK05	179	1520	173	1970	173	511
MK06	70	3317	64	3300	60	699
MK07	148	1778	144	2032	139	100
MK08	523	3007	523	6187	523	1118
MK09	341	5007	309	8525	307	1118
MK10	253	4613	205	7890	205	1460

Table 6. Comparison 2 between the three developed PSO variants on Raspberry Pi B+.

Instance	Basic PSO $I = 1454$ $S = 55$		One-Level PSO [32] $I = 1454$ $S = 55$		E2L-PSO without Reduction $\boldsymbol{I_1} = 64$ $\boldsymbol{S_2} = 50$ $\boldsymbol{I_2} = 5$ $\boldsymbol{S_2} = 5$	
	MS	CPU Time	*MS*	CPU Time	*MS*	CPU Time
4 × 5	12	16.08	11	7.2	11	4.14
8 × 8	19	40.51	16	36.71	14	9.54
10 × 10	12	71.06	10	79.61	7	11.40
MK01	44	64.60	40	122.17	39	25.84
MK02	33	83.98	29	133.38	27	24.44
MK03	204	826.5	204	579.69	204	81.52
MK04	69	174	62	217.33	60	49.59
MK05	177	275.95	173	303.14	173	62.51
MK06	78	658.2	64	607.62	60	144.20
MK07	148	264.86	144	310.27	139	49.42
MK08	523	569.88	523	1107.04	523	196.27
MK09	341	891.18	309	1290.77	307	197.86
MK10	252	974.16	205	1300.75	205	258.07

Table 7. Comparison 3 between the three developed PSO variants on STM32 F407VGT6.

Instance	Basic PSO $I = 100$ $S = 50$		One-Level PSO [32] $I = 100$ $S = 50$		E2L-PSO without Reduction $\boldsymbol{I_1} = 20$ $\boldsymbol{S_2} = 10$ $\boldsymbol{I_2} = 5$ $\boldsymbol{S_2} = 5$	
	MS	CPU Time	*MS*	CPU Time	*MS*	CPU Time
4×5	12	4.25	12	1.87	11	1.30
8×8	22	12.56	18	12.31	16	3.12
10×10	12	27.06	12	19.56	9	3.25
MK01	44	23.68	42	50.93	39	6.31
MK02	33	2.06	30	48	29	8.75
MK03	204	206.37	204	194.34	204	27
MK04	72	49.93	64	69.06	62	17.5
MK05	179	95	174	123.12	173	31.93
MK06	70	207.31	64	203.25	62	43.68
MK07	151	110.80	144	127	139	12.56
MK08	523	188	523	386.68	523	62.50
MK09	341	312.99	309	520.81	309	69.87
MK10	254	288.31	205	490.12	205	91.25

Table 8. Comparison 4 between the three developed PSO variants on Raspberry Pi B+.

Instance	Basic PSO $I = 100$ $S = 50$		One-Level PSO [32] $I = 100$ $S = 50$		E2L-PSO without Reduction $\boldsymbol{I_1} = 20$ $\boldsymbol{S_2} = 10$ $\boldsymbol{I_2} = 5$ $\boldsymbol{S_2} = 5$	
	MS	CPU Time	*MS*	CPU Time	*MS*	CPU Time
4×5	12	1.05	11	0.45	11	0.25
8×8	19	2.53	16	2.15	14	0.59
10×10	12	4.44	10	1.07	7	0.71
MK01	44	4.03	40	5.78	39	1.61
MK02	33	5.59	29	6.74	27	1.52
MK03	204	45.91	204	34.09	204	5.09
MK04	69	9.87	62	11.58	60	3.09
MK05	177	17.24	173	17.94	173	3.90
MK06	78	41.13	64	37.97	60	9.01
MK07	148	13.94	144	16.37	139	3.08
MK08	523	29.99	523	49.13	523	12.26
MK09	341	55.69	309	67.93	307	12.36
MK10	252	60.88	205	81.29	205	16.12

6.2.2. Experiment #2 (Adaptive E2L-PSO on Raspberry Pi B+)

To better visualize the results, we increase the number of visited points to 3,000,000 ($I_1 = 100$ $S_2 = 200$ $I_2 = 10$ $S_2 = 10$).

Table 9 presents the simulation results of the E2L-PSO without particle reduction on Raspberry Pi B+ card. In this table, the number of visited solutions is fixed at 3,000,000 points (we widen the search space) to better compare the quality of the results. This table is used only for comparison reasons. In this table, we can see that our algorithm reaches the lower bound values in three instances (bold values) compared to values presented in Table 4. For the rest of the instances, E2L-PSO gives comparable results.

Table 9. E2L-PSO without central processing unit (CPU) reduction needed (number of visited points is 3,000,000) on Raspberry Pi B+.

	MS	CPU
MK01	39	50.28
MK02	**26**	144.1
MK03	**204**	167.4
MK04	60	91.25
MK05	170	125.02
MK06	62	252.35
MK07	139	94.15
MK08	**523**	372.21
MK09	307	387.36
MK10	205	451.08

The results in Tables 10–12 are determined under this assumption: reduction needed requests of 90%, 40% and 20%, respectively, occur every 20 s after the first request_Period. These tables present the simulation results of the E2L-PSO with hard, medium and soft particle reduction on the Raspberry Pi B+ card.

Table 10. E2L-PSO hard CPU reduction needed (60%) on Raspberry Pi B+.

	Beginning 20%		Middle (50%)		End (90%)	
	MS	CPU	*MS*	CPU	*MS*	CPU
MK01	40	20.06	40	35.2	39	50.28
MK02	28	37.66	28	80.05	26	129.60
MK03	204	98.23	204	118.3	204	150.30
MK04	62	67.60	60	78.12	60	82.07
MK05	172	70.33	172	97.6	170	114.31
MK06	64	110.18	64	186.17	62	225.01
MK07	140	42.87	140	70.86	139	80.15
MK08	523	205.63	523	279.44	523	334.8
MK09	309	199.15	307	283.65	307	346.26
MK10	205	220.44	205	332.45	205	410.99

Table 11. E2L-PSO medium CPU reduction needed (40%).

	Beginning 20%		Middle (50%)		End (90%)	
	MS	CPU	*MS*	CPU	*MS*	CPU
MK01	39	37.71	39	48.10	39	46.19
MK02	28	62.90	27	99.17	26	139.50
MK03	204	111.60	204	133.80	204	158.02
MK04	60	72.03	60	81.52	60	88.17
MK05	172	90.60	171	100.87	170	120.03
MK06	64	140.17	62	191.98	62	238.40
MK07	140	63.34	140	81.55	139	87.81
MK08	523	241.70	523	299.02	523	353.11
MK09	309	264.98	307	318.47	307	361.74
MK10	205	282.53	205	374.67	205	439.60

Table 12. E2L-PSO soft CPU reduction needed (20%).

	Beginning 20%		Middle (50%)		End (90%)	
	MS	CPU	*MS*	CPU	*MS*	CPU
MK01	39	40.50	39	44.33	39	48.70
MK02	26	73.15	26	130.66	26	142.55
MK03	204	93.82	204	148.02	204	161.21
MK04	60	81.77	60	86.12	60	90.01
MK05	170	100.03	170	111.64	170	123.33
MK06	62	188.4	62	207.01	62	249.46
MK07	139	70.48	139	84.21	139	91.63
MK08	523	260.70	523	300.65	523	368.01
MK09	307	282.24	307	344.89	307	381.37
MK10	205	310.70	205	399.57	205	446.99

From Table 10, we note that compared to the results in Table 9:

- Case 1: E2L-PSO gives acceptable results for *MS* with 26% to 70% of CPU time gain.
- Case 2: E2L-PSO gives acceptable results for *MS* and best *MS* for 50% of benchmarks. E2L-PSO gives 15% to 44% of CPU time gain.
- Case 3: E2L-PSO gives the best results for *MS* in all benchmarks, with 10% of CPU time gain.

From Table 11, we note the following in comparison to the results in Table 9:

- Case 1: E2L-PSO gives acceptable results for *MS* and best *MS* for 60% of benchmarks. A 21% to 66% of CPU time gain is obtained.
- Case 2: E2L-PSO gives acceptable results for *MS* and best *MS* for 80% of benchmarks. An 11% to 30% of CPU time gain is obtained.
- Case 3: E2L-PSO gives best results for *MS* in all benchmarks, with 4% to 5% of CPU time gain.

From Table 12, we note that, compared to the results in Table 9:

- Case 1: E2L-PSO gives the best *MS* for 100% of benchmarks. A 10% to 49% of CPU time gain is obtained.
- Case 2: E2L-PSO gives the best *MS* for 100% of benchmarks with 6% to 9% of CPU time gain.
- Case 3: E2L-PSO gives the best results for *MS* in all benchmarks, with a poor CPU time gain (1%).

To conclude, we can say that the E2L-PSO give the best MS results with reduction in CPU time according to the flexibility of the problems, the CPU reduction needed and the request period.

6.2.3. Experiment #3 (Stability Analysis of Adaptive E2L-PSO)

The results of the two-tailed Mann–Whitney tests, comparing the outcomes of the second example (Table 11: Adaptive E2L-PSO with medium reduction and E2L-PSO without CPU reduction needed), are shown in Table 13.

The symbols U and p are the calculation of statistics, where U is the minimum value between the decisions variables of two groups and p is a non-parametric measure of the overlap between two distributions; it can take values between 0 and 1. We chose to work with a significance level = 0.05 and a number of samples (run) = 10, then the critical value of U at $p < 0.05$ is 23. To understand the results obtained, note that when the U-value increases (p increases), the difference between the two variants increases (i.e., high U-value = high similarity): $U = 0$ implies no similarity and $U = 50$ implies total similarity.

Table 13. Results of two-tailed Mann–Whitney tests.

	Beginning 20%		**Middle (50%)**		**End (90%)**	
	U	p	U	p	U	p
MK01	30	0.1415	40	0.4175	50	0.9681
MK02	0	0.0001	0	0.0001	50	0.9681
MK03	50	0.9681	50	0.9681	50	0.9681
MK04	10	0.0027	35	0.2713	50	0.9681
MK05	0	0.0001	0	0.0001	50	0.9681
MK06	0	0.0001	25	0.0643	50	0.9681
MK07	0	0.0001	0	0.0001	50	0.9681
MK08	50	0.9681	50	0.9681	50	0.9681
MK09	0	0.0001	25	0.0643	50	0.9681
MK10	50	0.9681	50	0.9681	50	0.9681

In the first case (Request_Period = 20%), the results show that there are total differences between the two variants on five benchmarks ($U = 0$ and $p < 0.05$), significant differences in MK04 ($U = 10 < 23$ and $p = 0.0027 < 0.05$), significant similarities in MK01 ($U = 30 > 23$ and $p = 0.1415 > 0.05$) and total similarities on four benchmarks.

In the second case (Request_Period = 50%), the results show that there are total differences between the two variants on three benchmarks ($U = 0$ and $p < 0.0001$), significant similarity on four benchmarks ($U > 23$ and $p > 0.05$) and a total similarity on three benchmarks.

In the third case (Request_Period = 90%), the results show that there are no differences (total similarities) between the two variants on all benchmarks ($U = 50$ and $p = 0.9681$).

6.2.4. Experiment #4 Robustness Analyses of Adaptive E2L-PSO

The results of the robustness measure comparing the outcomes of the second example (Table 11: adaptive E2L-PSO with medium reduction and E2L-PSO without CPU reduction needed) are shown in Table 14.

Table 14. Robustness analyses of E2L-PSO with medium CPU reduction needed (40%) on Raspberry Pi B+.

	Beginning 20%		**Middle (50%)**		**End (90%)**	
	MS	**RM**	***MS***	**RM**	***MS***	**RM**
MK01	39	0	39	0	39	0
MK02	28	10	27	6	26	5
MK03	204	0	204	0	204	0
MK04	60	1	60	1	60	1
MK05	172	12	171	11	170	10
MK06	64	6	62	4	62	4
MK07	140	14	140	14	139	11
MK08	523	0	523	0	523	0
MK09	309	24	307	21	307	21
MK10	205	0	205	0	205	0

- If RM < 0: risk of machine failure.
- If RM = 0: slack time of all machines equal to completion time. So, we can say that is acceptable robustness.
- If RM > 0: better robustness but there is a loss of time (conflicting with our objectives).

To resume, we can say that, in all the three scenarios and in the vast majority of cases, we have adequate robustness with improved makespan and shorter CPU times.

6.2.5. Experiment #5 (Previous Metaheuristics Are Compared)

Since we have not found any previous work running metaheuristic algorithms to solve FJSP on Raspberry Pi B+ or STM32 cards, we compare our results (E2L-PSO without reduction) with previous works executed in another environment (PCs). Experiments are conducted with 50 particles and 100 iterations (5000 visited points). Table 15 presents the comparison against PSO-based metaheuristics: PSO [35], artificial bee colony (ABC) [36], quantum annealing based optimization (QAO) [37], genetic algorithm (GA) [38], human learning optimization algorithm and PSO (HLO-PSO) [39] and hybrid brain storm optimization algorithm and late acceptance hill climbing (hybrid PSO) [40] (— denotes the data's unavailability). A bold value indicates that the E2L-PSO result is either optimal or the best. For all benchmarks, we note that the E2L-PSO performed on the Raspberry Pi B+ delivers the best makespan, except in two cases (MK02 and MK05), although the result is always close to the best of the alternatives.

Table 15. Comparison of the E2L-PSO against other works (non PSO-based and PSO-based metaheuristics).

	[35] PSO	[36] ABC	[37] QAO	[38] GA	[39] HLO-PSO	[40] Hybrid PSO	Our E2L-PSO	Our E2L-PSO
Machine	Intel Core i7; 16 GB RAM	Intel Core i3	n-a	Intel Core i5; 8 GB RAM	n-a	Intel Core i3; 4 GB RAM	Raspberry Pi B+ 1 GB RAM	STM32F407VGT6 192 KB
Particles	20	-	-	-	100	50	50	50
Iterations	50	-	-	-	500	200	100	100
4×5	11	11	-	11	11	-	**11**	**11**
8×8	14	14	-	14	14	-	**14**	16
10×10	-	11	-	11	7	-	**7**	9
MK01	40	40	41	40	40	40	**39**	**39**
MK02	29	27	27	27	28	**26**	27	29
MK03	204	204	204	204	204	204	**204**	**204**
MK04	66	60	67	60	63	-	**60**	62
MK05	175	172	176	**172**	175	173	173	173
MK06	77	63	62	69	71	61	**60**	62
MK07	145	140	144	144	144	141	**139**	**139**
MK08	523	523	523	523	523	523	**523**	**523**
MK09	320	307	314	320	326	307	**307**	309
MK10	239	214	214	254	238	204	**205**	**205**

7. Practical and Managerial Uses of E2L-PSO

Based on the findings obtained after testing on Raspberry Pi B+ and STM32 boards, we can conclude that our technique is preferable for handling optimization difficulties in embedded settings. The following are some examples of practical and managerial applications:

- Home staff scheduling is an example of a healthcare optimization problem.
- The difficulty of pattern recognition and image processing.
- Applications for detecting medical diseases.
- Problems with electrical power transmission optimization.
- Heterogeneous wireless sensor network routing and clustering.

All cyber–physical systems can support our technique E2L-PSO, which can be used to solve any NP-hard optimization issue.

8. Conclusions

In this work, we introduce an embedded adaptive two-level PSO (E2L-PSO) method for the FJSP that is based on the two level PSO algorithm [32]. The E2L-PSO approach was developed to be the standard variant dedicated to the dynamic environment. The experiments were conducted using our E2L-PSO algorithm on Raspberry Pi B+ and STM32F407VGT6. The results obtained show the efficiency of our metaheuristic method in adapting to the changes of the environment as well as to the CPU needs of the execution equipment.

Experiments run using our E2L-PSO algorithm on benchmark problems returned better results than those previously obtained by other metaheuristics in a much smaller amount of CPU time. The high quality of the solutions (MS) is due to the efficient explo-

ration of the solution space. The low CPU time is due to the use of the adaptive process (iteration reduction). The low machine load (workload) is assured by the appearance of the two-level notion. This is one of the goals achieved with the two-level PSO. All these criteria are ensured in real-time application.

In future work, we aim to include the memory reduction and the rescheduling method, developed for the breakdown cases, which should be included with E2L-PSO on an embedded system. We aim to apply this in the fields of sensor networks and IoT. To expand the capability of E2L-PSO, we should use additional machine learning approaches.

Author Contributions: Conceptualization, Methodology and Validation: R.Z., W.B.D. and S.M.; software, writing—original draft preparation, review and editing, R.Z. and S.M.; Supervision, A.J.; All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Deanship of Scientific Research, Qassim University.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Michael, G.R.; David, J.S.; Ravi, S. The complexity of flowshop and jobshop scheduling. *Math. Oper. Res*. **1976**, *1*, 117–129.
2. OrtÍz-Barrios, M.; Petrillo, A.; De Felice, F.; Jaramillo-Rueda, N.; Jimenez-Delgado, G.; Borrero-Lopez, L. A Dispatching-Fuzzy AHP-TOPSIS Model for Scheduling Flexible Job-Shop Systems in Industry 4.0 Context. *Appl. Sci*. **2021**, *11*, 5107. [CrossRef]
3. Calleja, G.; Rafael, P. A dispatching algorithm for flexible job-shop scheduling with transfer batches: An industrial application. *Prod. Plan. Control*. **2014**, *25*, 93–109. [CrossRef]
4. Chaudhry, I.A.; Khan, A.A. A research survey: Review of flexible job shop scheduling techniques. *Int. Trans. Oper. Res*. **2016**, *23*, 551–591. [CrossRef]
5. Krasimira, G.; Leoneed, K.; Vassil, G. A survey of solving approaches for multiple objective flexible job shop scheduling problems. *Cybern. Inf. Technol*. **2015**, *15*, 3–22.
6. Sangaiah, A.K.; Mohsen, Y.S.; Mehdi, S.; Seyed, M.B.; Hosseinabadi, A.; Ji, W. A new meta-heuristic algorithm for solving the flexible dynamic job-shop problem with parallel machines. *Symmetry* **2019**, *11*, 165. [CrossRef]
7. Park, J.-S.; Ng, H.Y.; Chua, T.-J.; Ng, Y.-T. Unified genetic algorithm approach for solving flexible job-shop scheduling problem. *Appl. Sci*. **2021**, *11*, 6454. [CrossRef]
8. Lin, C.R.; Zheng, C.C.; Meng, C.Z. Learning-Based Grey Wolf Optimizer for Stochastic Flexible Job Shop Scheduling. *IEEE Trans. Autom. Sci. Eng*. **2022**, 1–13. [CrossRef]
9. Tamssaouet, K.; Dauzere-Peres, S.; Knopp, S.; Bitar, A.; Yugma, C. Multiobjective optimization for complex flexible job-shop scheduling problems. *Eur. J. Oper. Res*. **2022**, *296*, 87–100. [CrossRef]
10. Nayak, S.; Sood, A.K.; Pandey, A. Integrated Approach for Flexible Job Shop Scheduling Using Multi-objective Genetic Algorithm. In *Advances in Mechanical and Materials Technology*; Springer: Singapore, 2022; pp. 387–395.
11. Xianghui, C.; Lu, L.; Wenlong, S.; Aurobinda, L.; Jin, T.; Yu, C. Real-time misbehavior detection and mitigation in cyber-physical systems over WLANs. *IEEE Trans. Ind. Inform.* **2015**, *13*, 186–197.
12. Kyoung-Dae, K.; Panganamala, K.R. Cyber–physical systems: A perspective at the centennial. *Proc. IEEE* **2012**, *100*, 1287–1308. [CrossRef]
13. Patricia, D.; Edward, L.A.; Sangiovanni, V.A. Modeling cyber–physical systems. *Proc. IEEE* **2011**, *100*, 13–28.
14. Hemalatha, R.; Prakash, R.; Sivapragash, C. Analysis on energy consumption in smart grid WSN using path operator calculus centrality based HSA-PSO algorithm. *Soft Comput. J*. **2019**, *24*, 1–13. [CrossRef]
15. Kartikeya, U.; Miroojin, B.; Vibhor, P.; Kumar, L.B. Distributed maintenance planning in manufacturing industries. *Comput. Ind. Eng*. **2017**, *108*, 1–14.
16. Venayagamoorthy, G.K.; Sheetal, D. Navigation of mobile sensors using PSO and embedded PSO in a fuzzy logic controller. In Proceedings of the IEEE Industry Applications Conference, 39th IAS Annual Meeting, Seattle, WA, USA, 3–7 October 2004; Volume 2, pp. 1200–1206.
17. Javier, R.L.; Christian, A.F.; Gregorio, T.P.; Rashid, M.; Joan, S.F. Enhancing metaheuristic-based online embedding in network virtualization environments. *IEEE Trans. Netw. Serv. Manag*. **2017**, *15*, 200–216.
18. Hereford, J.; Michael, S. Multi-robot search using a physically-embedded particle swarm optimization. *Int. J. Comput. Intell. Res*. **2008**, *4*, 197–209. [CrossRef]
19. Micael, S.C.; Patricia, V.A.; Rui, R.P.; Nuno, F.M.F. Benchmark of swarm robotics distributed techniques in a search task. *Robot. Auton. Syst*. **2014**, *62*, 200–213.

20. Alakananda, B.; Amit, K.; Swagatam, D.; Crina, G.; Ajith, A. Hardware software partitioning problem in embedded system design using particle swarm optimization algorithm. In Proceedings of the IEEE Intertional Conference on Complex, Intelligent and Software Intensive Systems, Washington, DC, USA, 4–7 March 2008; pp. 171–176.
21. Rezazadeh, H.; Ghazanfari, M.; Sadjadi, S.J.; Mir, B.; Aryanezhad, A.; Makui, A. Linear programming embedded particle swarm optimization for solving an extended model of dynamic virtual cellular manufacturing systems. *J. Appl. Res. Technol*. **2009**, *7*, 83–108. [CrossRef]
22. Hereford, J.M.; Siebold, M.A. Bio-inspired search strategies for robot swarms. In *Swarm Robotics from Biology to Robotics*; IntechOpen: London, UK, 2010.
23. Sheikholeslami, R.; Kaveh, A. A survey of chaos embedded meta-heuristic algorithm. *Int. J. Optim. Civ. Eng*. **2013**, *3*, 617–633.
24. Kamlesh, M.; Li, Z.; Siew, N.C.; Chee Peng, L.; Ben, F. A micro-GA embedded PSO feature selection approach to intelligent facial emotion recognition. *IEEE Trans. Cybern*. **2016**, *47*, 1496–1509.
25. Qi, X.; Li, Z.; Chen, C.; Liu, L. A wireless sensor node deployment scheme based on embedded virtual force resampling particle swarm optimization algorithm. *Appl. Intell*. **2021**, 1–22. [CrossRef]
26. Özdemir, M.T. Optimal parameter estimation of polymer electrolyte membrane fuel cells model with chaos embedded particle swarm optimization. *Int. J. Hydrog. Energy* **2021**, *46*, 16465–16480. [CrossRef]
27. Shariati, M.; Mafipour, M.S.; Mehrabi, P.; Alireza, B.; Yousef, Z.; Musab, N.A.S.; Hoang, N.; Jie, D.; Xuan, S.; Shek, P.-N. Application of a hybrid artificial neural network-particle swarm optimization (ANN-PSO) model in behavior prediction of channel shear connectors embedded in normal and high-strength concrete. *Appl. Sci*. **2019**, *9*, 5534. [CrossRef]
28. Zhou, N.R.; Luo, A.W.; Zou, W.P. Secure and robust watermark scheme based on multiple transforms and particle swarm optimization algorithm. *Multimed. Tools Appl*. **2019**, *78*, 2507–2523. [CrossRef]
29. Qu, C. Virtual reconstruction of random moving image capturing points based on chaos embedded particle swarm optimization algorithm. *Microprocess. Micro-Syst*. **2020**, *75*, 103069. [CrossRef]
30. Vijh, S.; Sharma, S.; Gaurav, P. Brain tumor segmentation using OTSU embedded adaptive particle swarm optimization method and convolutional neural network. In *Data Visualization and Knowledge Engineering*; Springer: Berlin/Heidelberg, Germany, 2020; Volume 32, pp. 171–194.
31. Kacem, I.; Hammadi, S.; Borne, P. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Trans. Syst. Man Cybern. Part (Appl. Rev.)* **2002**, *32*, 1–13. [CrossRef]
32. Zarrouk, R.; Bennour, I.E.; Jemai, A. A two-level particle swarm optimization algorithm for the flexible job shop scheduling problem. *Swarm Intell*. **2019**, *13*, 145–168. [CrossRef]
33. Brandimarte, P. Routing and scheduling in a flexible job shop by tabu search. *Ann. Oper. Res. J*. **1993**, *41*, 157–183. [CrossRef]
34. Dennis, B.; Josef, G.M. *Test Instances for the Flexible Job Shop Scheduling Problem with Work Centers*; Research Report RR-12-01-01; Institut für Betriebliche Logistik und Organisation Arbeitspapier: Hamburg, Germany, 2012.
35. Ding, H.; Xingsheng, G. Improved particle swarm optimization algorithm based novel encoding and decoding schemes for flexible job shop scheduling problem. *Comput. Ind. Eng*. **2020**, *121*, 104951. [CrossRef]
36. Caldeira, H.R.; Gnanavelbabu, A.; Solomon, J.J. Solving the Flexible Job Shop Scheduling Problem Using a Hybrid Artificial Bee Colony Algorithm. In *Trends in Manufacturing and Engineering*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 833–843.
37. Denkena, B.; Schinkel, F.; Pirnay, J.; Wilmsmeier, S. Quantum algorithms for process parallel flexible job shop scheduling. *CIRP J. Manuf. Sci. Technol*. **2021**, *33*, 100–114. [CrossRef]
38. Chen, R.; Yang, B.; Li, S.; Wang, S. A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Comput. Ind. Eng*. **2020**, *149*, 106778. [CrossRef]
39. Ding, H.; Gu, X. Hybrid of human learning optimization algorithm and particle swarm optimization algorithm with scheduling strategies for the flexible job-shop scheduling problem. *Neurocomputing* **2020**, *414*, 313–332. [CrossRef]
40. Alzaqebah, M.; Jawarneh, S.; Alwohaibi, M.; Alsmadi, M.K.; Almarashdeh, I.; Mohammad, R.M.A. Hybrid Brain Storm Optimization algorithm and Late Acceptance Hill Climbing to solve the Flexible Job-Shop Scheduling Problem. *J. King Saud-Univ.-Comput. Inf. Sci*. 2020, *in press*. [CrossRef]