TOP (2021) 29:694–721
https://doi.org/10.1007/s11750-020-00588-5

ORIGINAL PAPER

Multi-objective scheduling on two dedicated processors

Adel Kacem[1] · **Abdelaziz Dammak**[1]

Received: 2 May 2019 / Accepted: 20 November 2020 / Published online: 3 January 2021

Abstract
We study a multi-objective scheduling problem on two dedicated processors. The aim is to minimize simultaneously the makespan, the total tardiness and the total completion time. This NP-hard problem requires the use of well-adapted methods. For this, we adapted genetic algorithms to multi-objective case. Four methods are presented to solve this problem. The first is an aggregative genetic algorithm (GA), the second is a Pareto GA, the third is a non-dominated sorting GA (NSGA-II) and the fourth is a constructive algorithm based on lower bounds (CABLB). We proposed some adapted lower bounds for each criterion to evaluate the quality of the found results on a large set of instances. Indeed, these bounds also make it possible to determine the dominance of one algorithm over another based on the different results found by each of them. We used two metrics to measure the quality of the Pareto front: the hypervolume indicator (HV) and the number of solutions in the Pareto front (ND). The obtained results show the effectiveness of the proposed algorithms.

Keywords Scheduling · Dedicated processors · Lower bound · NSGA-II · Pareto front

Mathematics Subject Classification 90-08

✉ Adel Kacem
adel.kacem@gmail.com

Abdelaziz Dammak
abdelaziz.dammak@fsegs.rnu.tn

[1] Modeling and Optimization for Decisional, Industrial and Logistic Systems Laboratory, Faculty of Economics and Management, University of Sfax, Airport Street, km 4, Post Office Box 1088, 3018 Sfax, Tunisia

1 Introduction

This work aims at optimizing the computer control systems when these systems have two dedicated processors: the assignment of tasks to these processors is fixed. For this problem, we have three types of tasks. Some tasks must be processed only by the first processor, others by the second processor, and the remaining tasks need simultaneously both processors. This problem represents a practical issue in computer control systems, where a task is performed in several copies on different processors to ensure better safety of the system. In production management, we can cite the case where a task requires several operators for its execution.

The contribution of our work is to propose lower bounds for the three studied criteria (makespan, total tardiness and total completion time) and to develop genetic algorithms to solve this problem in the multi-objective case. The lower bounds allow us to assess the quality of the feasible solutions and the genetic algorithms incorporates the optimization part. We implemented our approach by considering aggregative, NSGA-II and Pareto scenarios on a large set of instances. The results show the effectiveness of the implemented algorithms. The studied problem is $P2|fix_j, r_j|C_{\max}, \sum T_j, \sum C_j$ according the standard ternary notation, where $P2$ denotes two processors; fix_j indicates that each task has one or two dedicated processors and the assignment of each task is fixed; r_j denotes the release date; $C_{\max}$, $\sum T_j$ and $\sum C_j$ indicates the makespan, the total tardiness and the total completion time, respectively.

The next section is a review of existing research related to the studied problem. In Sect. 3, a mathematical formulation model is proposed, some notations are detailed and the proposed lower bounds for the makespan, total completion time and total tardiness are given. In Sect. 4, we present the solving approaches. Four methods to solve the considered problem are developed. The first one is aggregative GA with Uniform Design, the second is Pareto GA, the third is the NSGA-II and the fourth is CABLB algorithm. Section 5 deals with the generation of instances, the different parameters used to develop the proposed algorithms, the proposed metrics to measure the quality of the Pareto front, the computational results and the qualitative and quantitative analysis. Finally concluding remarks are given in Sect. 6.

2 Literature review

Few studies have dealt with this problem. The most important studies are mentioned in the following paragraphs.

Coffman et al. (1985) studied the file transfer problem in the field of computer networks where each computer has a number of different ports for data exchange. File transfer uses a subset of ports, therefore, a multiprocessor task on dedicated processors. The boot time of the transfers is also taken into account, then different transfer protocols are proposed, and performance results are demonstrated. Drozdowski (1996) cited this paper to describe the actual applications of scheduling problems on dedicated processors.

Craig et al. (1988) studied the problem in testing integrated circuits VLSI (very large-scale integration). To test a component of these circuits, several other electronic components are needed simultaneously. The authors addressed the problems in case when the processing times are unitary or arbitrary. A heuristic based on the maximum degree of incompatibility has been proposed to solve these two problems $P|fix_j, p_j = 1|C_{\max}$ and $P|fix_j|C_{\max}$ (p_j denotes the processing time of task j).

These works (Coffman et al. 1985; Craig et al. 1988), which cover several fields of application, made it possible to form the first theoretical basis for scheduling problems on dedicated processors. This topic has been widely investigated during the past years. The most remarkable work has been devoted to the study of the complexity.

Hoogeveen et al. (1994) showed that the problem $P2|fix_j|\sum w_jC_j$ is NP-hard in the strong sense (w_j the weight and C_j the completion time of task j). The preemption of tasks does not make the problem easier. Oguz and Ercan (2005) proved that the problem $P2|fix_j, pmtn|\sum w_jC_j$ is NP-hard in the strong sense ($pmtn$ allows us the preemption of tasks). Afrati et al. (1999) proposed a polynomial time approximation scheme (PTAS) for the problem $Pm|fix_j, pmtn|\sum C_j$ and a second PTAS approximation scheme proposed by Afrati and Milis (2006).

Chu (1992) proposed a lower bound for the minimization of total tardiness problem; the calculation involves the SRPT priority rule (Shortest Remaining processing Time) for a relaxed problem with preemption. The main idea is that each time the processor becomes available, an unfinished task available with the shortest remaining processing time is set. The execution of a task is interrupted when its remaining processing time is strictly greater than the length of processing task that becomes available.

Leung and Wang (2000) proposed a genetic algorithm with multiple fitness functions to conduct research to solve a multi-objective problem. The authors applied an experimental design method called Uniform Design to select the weights used with the objective functions and diversify uniformly selected solutions.

Kacem (2007) developed two lower bounds for tardiness minimization problem on a single machine with Family Setup Times. The first lower bound is based on Emmons theorem (Emmons 1969) and the SPT rule (Shortest Processing Time), the second is achieved by sorting tasks by processing times and the idea of due dates exchange. Another idea for solving the linear programming problem was also proposed.

Berrichi et al. (2007) studied a bi-objective model of parallel machine problem using reliability models to take into account the service side. Two genetic algorithms were developed to obtain an approximation of the Pareto front: One algorithm that uses the two objectives weighted and NSGA-II algorithm.

Rebai et al. (2010) introduced three lower bounds for minimization tardiness problem on one machine to schedule preventive maintenance tasks. The first lower bound is based on the Lagrangian relaxation of mathematical model. The second is obtained by the sum of M costs calculated for M tasks, and the third is an adaptation of the lower bound given by Li (1997) for the problem of earliness tardiness minimization with a single due date for each task.

Manaa and Chu (2010) proposed a branch-and-bound method to minimize the makespan. In their article, the authors presented a lower bound that has been proven. This method can treat all instances generated up to 30 tasks for the most difficult cases in less than 15 min.

Vallada and Ruiz (2011) studied the unrelated parallel machine scheduling problem. A genetic algorithm is developed to solve this problem. The proposed method includes a fast local search and a local search enhanced crossover operator. The computational and statistical analysis shows an excellent performance in a comprehensive benchmark set of instances.

Bradstreet (2011) introduced the hypervolume indicator (HV) to measure the quality of the Pareto front. The hypervolume is one of the most famous indicator that can reflect the dominance of Pareto fronts.

An approximation algorithm is proposed by Alhadi et al. (2020) to minimize the maximum lateness and makespan on parallel machines. In this paper, the authors presented polynomial time approximation schemes to generate an approximate Pareto Frontier.

Kacem and Dammak (2019) studied the problem of bi-objective scheduling of multi-processor tasks on two dedicated processors. The authors adapted the genetic algorithm to solve the problem of minimizing the makespan and the total tardiness for the large size instances. The results found showed the effectiveness of the proposed genetic algorithms and the encouraging quality of the lower bounds constructed in Manaa and Chu (2010), Kacem and Dammak (2017).

The most three main criteria analyzed in the case of scheduling problems with one processor and with parallel processors are the schedule length, the mean flow time and the lateness (Blazewicz et al. 2019). In addition, these criteria are completely different: the schedule length and the flow time involving release dates. The lateness involves due dates. For that, we decided to study a new extension of this problem with these three criteria since they are the most relevant. According the standard ternary notation, the studied problem is $P2|fix_j, r_j|C_{\max}, \sum T_j, \sum C_j$.

3 Problem statement

In this section, we detail some notations, we propose a mathematical model for our studied problem and we give a lower bounds for the three criteria: the makespan, the total tardiness and the total completion time.

3.1 Notation

The following fields used in the studied problem $P2|fix_j, r_j|C_{\max}, \sum T_j, \sum C_j$ denote:

- $P2$: Two processors.
- fix_j: Each task has one or two dedicated processors and the assignment of each task is fixed.
- r_j: Release date of task j.

- $C_{\max}$: Makespan.
- T_j: Tardiness of task j; $T_j = max\{C_j - d_j;0\}$ with d_j the due date of task j.
- C_j: the completion time of task j.

The set of parameters and variables necessary for our model are presented below:

- $x_{l,k} = \begin{cases} 1 & \text{if task } l\text{completes before task } k \text{ starts} \\ 0 & \text{else;} \end{cases}$
- p_j: Processing time of task j.
- P_1: The set of tasks requiring the first processor.
- P_2: The set of tasks requiring the second processor.
- $P_{1,2}$: The set of tasks requiring both processors simultaneously.
- M: Constant penalty.

3.2 Mathematical model

We propose here a mathematical formulation of our multi-objective scheduling problem.

$$\text{Minimize } \{C_{\max}, \sum T_j, \sum C_j\} \tag{1}$$

Subject to:

$$C_k \geq C_l + p_k + (x_{l,k} - 1).M \ \forall (l,k) \in (P_1 \cup P_{1,2})^2 \text{ with } l \neq k \tag{2}$$

$$C_k \geq C_l + p_k + (x_{l,k} - 1).M \ \forall (l,k) \in (P_2 \cup P_{1,2})^2 \text{ with } l \neq k \tag{3}$$

$$x_{l,k} + x_{k,l} = 1 \ \forall (l,k) \in (P_1 \cup P_{1,2})^2 \text{ with } l \neq k \tag{4}$$

$$x_{l,k} + x_{k,l} = 1 \ \forall (l,k) \in (P_2 \cup P_{1,2})^2 \text{ with } l \neq k \tag{5}$$

$$C_{\max} \geq C_j \ \forall j \in (P_1 \cup P_2 \cup P_{1,2}) \tag{6}$$

$$T_j \geq C_j - d_j \ \forall j \in (P_1 \cup P_2 \cup P_{1,2}) \tag{7}$$

$$T_j \geq 0 \ \forall j \in (P_1 \cup P_2 \cup P_{1,2}) \tag{8}$$

$$C_j \geq r_j + p_j \ \forall j \in (P_1 \cup P_2 \cup P_{1,2}) \tag{9}$$

$$x_{k,l} \in \{0,1\} \ \forall k \neq l \ \forall (k,l) \in ((P_1 \cup P_{1,2})^2 \cup (P_2 \cup P_{1,2})^2) \tag{10}$$

Equation (1) expresses the three criteria: the makespan, the total tardiness and the total completion time. In the constraints (2) and (3), for each task k sequenced after

task l completes, the completion time of task k is greater than or equal to the completion time of task l plus the processing time of task k. The constraints (4) and (5) make it possible two tasks to complete one before the second task starts. The constraint (6) ensures that the makespan is greater than or equal to the completion time for each task j. The constraints (7) and (8) compute the tardiness and ensures that $T_j = max\{C_j - d_j;0\}$. Constraint (9) defines the completion time and shows that it is greater than or equal to the release date plus the processing time of task j. The constraint (10) defines the domain of definition of the parameters of the model.

We study three scheduling problems (makespan, total tardiness and total completion time) on two dedicated processors. To assess the quality of the results found by such a method, we use the following lower bounds for each criterion.

3.3 Lower bound *LBC* for problem $P2|fix_j, r_j|C_{max}$

Manaa and Chu (2010) proposed two ideas to construct a lower bound for the considered problem:

- The idea of dividing the problem into two sub-problems on one processor by relaxing the studied problem.
- The idea of Bianco et al. (1997) an optimal solution to minimize the makespan for one-processor problem.

The relaxation of the studied problem allows us to obtain two simple problems:

(a) Scheduling tasks that necessitate using simultaneously both processors and tasks that require the first processor.
(b) Scheduling tasks that require employing simultaneously both processors and tasks that necessitate the second processor.

The optimal solutions of problems (a) and (b) can be found by scheduling tasks according to the order of their release dates.

The lower bound for the studied problem corresponds to the maximum value of the solutions of problems (a) and (b).

3.4 Lower bound *LBTC* for problem $P2|fix_j, r_j|\sum C_j$

In this study, we use and combine three ideas to build a lower bound:

– The idea of reducing the problem into two sub-problems on one processor by partitioning the bi-processor tasks.
– The idea of dividing the mono- processor tasks into two tasks.
– The idea of under-estimating the completion times of the tasks on a single processor (originally proposed by Chu 1992).

The first of this lower bound is to partition the bi- processor tasks into two mono-processor tasks, each of them on one of the processors. We get two independent problems on each processor.

On the first processorP_1, we consider the n_1 mono-processor tasks j_j with a weight $w_j^1 = 1$, and the n_{12} bi- processor sub-tasks j_j on processor P_1 a weight $w_j^1 = w$, with $w \in [0, 1]$.

Similarly, we consider the second processor P_2 , the n_2 mono-processor tasks j_j with a weight $w_j^2 = 1$. However, the bi-processor sub-tasks j_j on processor P_2 a weight $w_j^2 = 1 - w$. Thus, we obtain a problem on each processor: $1|fix_j, r_j|\sum w_j^1 C_j$ and $1|fix_j, r_j|\sum w_j^2 C_j$.

We consider $w = \frac{1}{2}$. The next step is to divide the mono-processor tasks (with a weight $w_j^1 = 1$) in two tasks. We get for each divided task two sub tasks j_{j1} and j_{j2} with release date $r_{j1} = r_j$; $r_{j2} = r_j + \frac{p_j}{2}$ and processing time $p_{j1} = p_{j2} = \frac{p_j}{2}$.

We divide the weight on for each sub tasks. We are getting $w_j^1 = \frac{1}{2}$ if $\forall j_j \in \{P_1; P_{12}\}$. From where, $Lb1 = Lb\Big(1|fix_j, r_j|\sum w_j^1 C_j\Big) = Lb\Big(1|fix_j, r_j|\sum \frac{1}{2} C_j\Big) + \sum_{j\in P_1} \frac{p_j}{4}$, with $\sum_{j\in P_1} \frac{p_j}{4}$ is a penalty to be added according to Webster formula.

$$Lb1 = Lb\Big(1|fix_j, r_j|\sum \frac{1}{2} C_j\Big) + \sum_{j\in P_1} \frac{p_j}{4} = \frac{1}{2} Lb\Big(1|fix_j, r_j|\sum C_j\Big) + \sum_{j\in P_1} \frac{p_j}{4}.$$

We apply the same principle for the problem $1|fix_j, r_j|\sum w_j^2 C_j$. We are getting $Lb2 = Lb\Big(1|fix_j, r_j|\sum \frac{1}{2} C_j\Big) + \sum_{j\in P_2} \frac{p_j}{4} = \frac{1}{2} Lb\big(1|fix_j, r_j|\sum C_j\big) + \sum_{j\in P_2} \frac{p_j}{4}$.

$LBTC = Lb1 + Lb2$ is then a lower bound for problem $P2|fix_j, r_j|\sum C_j$. The calculation of the lower bounds of completion times for the problem on each processor uses the following theorem for $1|r_j, pre|\sum C_j$.

Theorem 1 (Chu 1992) *Let $C_{[i]}(\sigma)$ be the completion time of the task in the ith position of a feasible schedule. C'_i is the completion time of the task in the i th position of a feasible schedule constructed by the SRPT (Shortest Remaining Processing Time) priority rule. Chu proved that for every feasible schedule σ, we have: $C_{[i]}(\sigma) \geq C'_i$*

Table 1 Example

j	r_j	p_j	P
j_1	2	6	P_1
j_2	4	2	P_1
j_3	1	2	P_{12}
j_4	0	8	P_1
j_5	3	2	P_2
j_6	2	6	P_2
j_7	1	2	P_2

Table 2 Results division on P_1

j	r_j	p_j	P
$j_{1.1}$	2	3	P_1
$j_{1.2}$	5	3	P_1
$j_{2.1}$	4	1	P_1
$j_{2.2}$	5	1	P_1
j_3	1	2	P_1
$j_{4.1}$	0	4	P_1
$j_{4.2}$	4	4	P_1

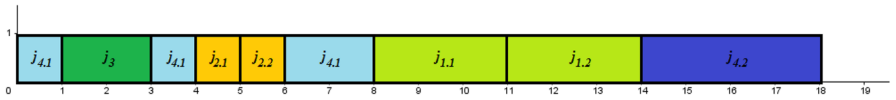

Fig. 1 SRPT representation on P_1

By applying the theorem in Chu (1992), we compute a lower bound on the completion time of each job.

Example Let us consider the instance in Table 1.

We apply the principle of calculation of the lower bound mentioned above and we get two sub-problems on each processor. On the first processor P_1, we schedule the tasks $\{j_1, j_2, j_3, j_4\}$. We divide the tasks $\{j_1, j_2, j_4\}$ on into two, we get the following tasks: $\{j_{11}, j_{12}, j_{21}, j_{22}, j_{41}, j_{42}\}$with the following parameters described in Table 2.

The sequence built by the SRPT rule with preemption gives the solution described in Fig. 1.

The total of completion time, giving the following lower bound: $Lb1 = \frac{1}{2}Lb\left(1|fix_j, r_j|\sum C_j\right) + \sum_{j\in P_1}\frac{p_j}{4} = \frac{3+5+6+8+11+14+18}{2} + \frac{3+3+1+1+4+4}{4} = 36,5$. Respectively, we calculate the lower bound $Lb2$ for the problem on P_2.

Thus, we consider $LBTC = Lb1 + Lb2$ as a lower bound for the problem $P2|fix_j, r_j|\sum C_j$. .

3.5 Lower bound *LBTT* for the problem $P2|fix_j, r_j|\sum T_j$

Kacem and Dammak (2017) proposed an adapted lower bound for the problem of minimization of total tardiness on two dedicated processors. The authors exploited and combined three ideas to construct this lower bound:

– The idea of reducing the problem in two sub-problems on one processor by partitioning the bi-processor tasks.
– The idea of under-estimating the completion time of the tasks (initially suggested by Chu 1992).
– The idea of calculating the lower bound by assigning the due dates to the reduced completion times (originally proposed in Rebai et al. (2010) for another scheduling problem).

The first step of this lower bound is to divide the bi-processor tasks into two mono-processor tasks; each of which is executed on one of the two processors. Consequently, we obtain two independent problems on each processor. On the first processor P_1, we consider the n_1 mono-processor tasks J_j with a weight $\lambda_j^1 = 1$, and the n_{12} bi-processor sub-tasks J_j on processor P_1 having a weight $\lambda_j^1 = \lambda$ with $\lambda \in [0, 1]$.

Similarly, we consider, on the second processor P_2, the n_2 mono-processor tasks J_j with a weight $\lambda_j^2 = 1$. However, the n_{12} bi-processor sub-tasks J_j on the processor P_2 have a weight $\lambda_j^2 = 1 - \lambda$. Thus, we obtain a problem on each processor $1|fix_j, r_j|\sum \lambda_j^1 T_j$ and $1|fix_j, r_j|\sum \lambda_j^2 T_j$.

Using the idea of Chu (1992) (described in the previous section), we compute a lower bound on the completion time of each task.

The next step of computing the lower bound is based on the idea of assigning the weight and the due date of each task to completion times' lower bounds. The total tardiness is minimized by the Hungarian algorithm.

Let $Cost_{i,j}$ be the cost of assigning a reduced C_i' to the task J_j supposed to end at the i^{th} position of the schedule. This cost can be calculated according to the following formula: $Cost_{i,j} = \lambda_j * max\{C_i' - d_j; 0\}$.

This assignment technique, presented by Rebai et al. (2010), allows us to elaborate a new lower bound. We apply the Hungarian algorithm to determine, from the assignment matrix $Cost_{i,j}^{P_1}$, a lower bound ($Lb1$) to solve the following problem $1|fix_j, r_j|\sum \lambda_j^1 T_j$.

$$Lb1 = min \sum x_{i,j} Cost_{i,j} \quad \begin{cases} \sum_i x_{i,j} = 1 \\ \sum_j x_{i,j} = 1 \\ x_{i,j} \in \{0, 1\} \end{cases} \tag{11}$$

Applying the same process, we calculate $Lb2$ for the $1|fix_j, r_j|\sum \lambda_j^2 T_j$ problem. Thus, we consider $LBTT = Lb1 + Lb2$ as a lower bound for the problem $P2|fix_j, r_j|\sum T_j$.

Optimization of the lower bound *LBTT* To improve the constructed lower bound, we look for the weights λ_j^*, which maximize *LBTT*. The idea is to associate the better weight λ_j^* for each bi-processor task J_j which maximizes the tardiness calculated by the Hungarian algorithm. We use the following method to optimize the bound *LBTT*.

- For a bi-processor task $J_j \in (P_{1,2})$, we calculate the gap $e_j *$ between $T_{j^*}^{P_1}$ and $T_{j^*}^{P_2}$ (where $T_{j^*}^{P_1}$ (respectively, $T_{j^*}^{P_2}$) is the tardiness of task j associated to the sub-problem on P_1 (respectively, P_2) obtained by the Hungarian algorithm).
- According to the gap, $e_j *$, we increase the λ value for a negative gap (and we reduce it, respectively, for a positive gap).
- We apply the Hungarian algorithm to the new matrix and we calculate a new lower bound $LBTT^{\lambda}$.
- We repeat this procedure $\forall J_j \in (P_{1,2})$.

Next, we present the study of the problem $P2|fix_j, r_j|C_{\max}, \sum C_j, \sum T_j$.

4 Solving approaches

We adapt a genetic algorithm to the multi-objective case. We propose four algorithms to solve the considered problem. The first is an aggregative GA, the second is a Pareto GA, the third is a NSGA-II and the fourth is a constructive algorithm based on lower bounds.

4.1 The genetic algorithm

To represent the data of the studied problem, we used a standard coding technique. This coding consists in representing an individual with a permutation containing N distinct numbers that correspond to the set $\{1, 2, 3, .., N\}$.

To form the diversified initial population, we used a random method to create a feasible sequence and to generate the other individuals of the initial population.

To assess the quality of individuals in a population, we have presented three methods to evaluate the studied problem in a given sequence.

The literature has several selection techniques such as proportional selection by tournament, by rank, random selection, etc (see Karasakal and Silav 2016). For our algorithm, we implemented three selection approaches: the aggregative approach, the Pareto one and NSGA-II.

The process of crossover between two parents leads to the birth of two children. In this case, an exchange position is randomly determined (see Vallada and Ruiz 2011). The first part of the first child is directly obtained from the first parent. The second part is provided by respecting the order of the remaining tasks as they appear in the second parent tasks. The same process is applied to the second child by reversing the parents. For our algorithm, we implemented the one-point crossover, which is a folklore (see Holland 1975).

Several methods of mutation exist in the literature such as the method of permutation, insertion and inversion. In our case, we used the permutation method of swapping two positions of the individual.

4.2 Aggregative approach

To adapt our genetic algorithm to the multi-objective case, we constructed an aggregative selection method that consists in generating weights for each sequence of a given population. To calculate the weights, we used an experimental design method called Uniform Design (Leung and Wang 2000). We choose a new population by a scaling method, which consists in calculating the weighted sum of normalized objective functions. Several combinations of weight are considered for the three objective functions (makespan, total tardiness and completion time). Each combination of these weights transforms the problem into a mono-objective case. Accordingly, the search directions are uniformly dispersed to the Pareto front in the objective space. With multiple fitness functions, we design a selection scheme to maintain the quality and the diversity of the population. This selection scheme consists in applying at every iteration the fitness functions (See Equation 19) and to sort the individuals of the current population in increasing order according to the Scaling method (See Sect. 4.2.2). The best individuals are selected to form a new population.

In what follows, we will describe the Uniform Design method used for calculating the weight and we will give the formula for the scaling method for the selection of a new population.

4.2.1 Calculation of weight with uniform design

The main objective of the Uniform Design is to sample a small set of points from a given set of points, so that the selected points are uniformly dispersed. This method is a branch of statistics that has been used to calculate the weight. As an illustration of the Uniform Design method, the reader could consult (Leung and Wang 2000).

We consider a unit hyper-cube C in a K dimensions space (K is the number of objectives) and h a point in C, Where $h = (h_1, h_2, .., h_K)^T$, such that $0 \leq h_i \leq 1$ $\forall 1 \leq i \leq K$.

For any item h from the hyper-cube C, we can create a hyper-rectangle $R(h)$ between the center O and h, with $O = (0, 0, .., 0)^T$. This hyper-rectangle is described by the following formula:

$$R(h) = \left\{ a \in C / a = (a_1, a_2, .., a_K), 0 \leq a_i \leq h_i, \forall 1 \leq i \leq K \right\} \tag{12}$$

We consider a set of X points from C, we can associate with each point h, a subset of X points that belongs to the hyper-rectangle $R(h)$. Let $X(h)$ be the cardinality of such a sub set and $X(h)/X$ the fraction of the points included in the hyper-cube C and $\prod_{i=1}^{K} h_i$ is the fraction of volume value of the hyper-rectangle $R(h)$. The uniform design is to determine X points in C such that the following discrepancy is minimized.

$$\underset{h\in C}{Sup}\left|\frac{X(h)}{X}-\prod_{i=1}^{K}h_i\right| \tag{13}$$

The authors presented the X points solution calculated using the uniform matrix $U(K,X)\left\{U_{r,i}\right\}_{X*K}$ given by Fang and Li (1994). With $U_{r,i}=(r.\sigma^{i-1}modX)+1$ and σ is a parameter that depends on X and K.

Now, we consider our problem studied, which consists in optimizing three objectives. In our case, we have $K=3$, we take $X=7$, so $\sigma=3$ see (Leung and Wang 2000). Using the formula $U_{r,i}$ given by Fang and Li (1994), we get the following uniform matrix:

$$U(3,7)\left\{U_{r,i}\right\}_{X*K}=\begin{pmatrix}2&4&3\\3&7&5\\4&3&7\\5&6&2\\6&2&4\\7&5&6\\1&1&1\end{pmatrix}$$

We consider the weighting vector $W^r=(w_1^r,w_2^r,..,w_K^r)^T$. The components of this vector are calculated by the following formula:

$$w_i^r=\frac{U_{r,i}}{U_{r,1}+U_{r,2}+\cdots+U_{r,K}},\forall 1\le r\le X,\forall 1\le i\le K \tag{14}$$

4.2.2 Scaling method

We use the weight components vector calculated using the Uniform Design to build a scaling method that allows us to choose a new population by sorting individuals of the current population in ascending order according to the following formula:

$$H(s)=w_1^r*C(s)+w_2^r*TT(s)+w_3^r*TC(s) \tag{15}$$

Such that,

$$C(s)=\left(\frac{C_{\max}(s)-\underset{x\in P}{min}\left\{C_{\max}(x)\right\}}{\underset{x\in P}{max}\left\{C_{\max}(x)\right\}-\underset{x\in P}{min}\left\{C_{\max}(x)\right\}}\right) \tag{16}$$

$$TT(s)=\left(\frac{T(s)-\underset{x\in P}{min}\{T(x)\}}{\underset{x\in P}{max}\{T(x)\}-\underset{x\in P}{min}\{T(x)\}}\right) \tag{17}$$

$$TC(s) = \left(\frac{C_{time}(s) - \min\limits_{x \in P}\{C_{time}(x)\}}{\max\limits_{x \in P}\{C_{time}(x)\} - \min\limits_{x \in P}\{C_{time}(x)\}} \right) \tag{18}$$

where w_i^r are the vector components of the weight described in the above section, s is a feasible solution from population P and $C_{\max}(x)$, $T(x)$, $C_{time}(x)$, are, respectively, the makespan, the total tardiness and total completion time of a solution $x \in P$.

By exploiting the uniform matrix, we obtain seven evaluation functions (fitness). The list of functions is given by the following formula:

$$\begin{cases} fitness_1 = \frac{2}{9} * C(s) + \frac{4}{9} * TT(s) + \frac{3}{9} * TC(s) \\ fitness_2 = \frac{3}{15} * C(s) + \frac{7}{15} * TT(s) + \frac{5}{15} * TC(s) \\ fitness_3 = \frac{4}{14} * C(s) + \frac{3}{14} * TT(s) + \frac{7}{14} * TC(s) \\ fitness_4 = \frac{5}{13} * C(s) + \frac{6}{13} * TT(s) + \frac{2}{13} * TC(s) \\ fitness_5 = \frac{6}{12} * C(s) + \frac{2}{12} * TT(s) + \frac{4}{12} * TC(s) \\ fitness_6 = \frac{7}{18} * C(s) + \frac{5}{18} * TT(s) + \frac{6}{18} * TC(s) \\ fitness_7 = \frac{1}{3} * C(s) + \frac{1}{3} * TT(s) + \frac{1}{3} * TC(s) \end{cases} \tag{19}$$

Each combination of these weights transforms the problem into a mono-objective case. For each combination, the genetic algorithm is applied simultaneously, the best feasible solutions are selected to form a new population by sorting the individuals of the current population in increasing order of the weighted objective and the population is stored. At the end of this process, such populations are merged and only the non-dominated solutions are kept.

4.3 Pareto approach

We adapt the classical genetic algorithm for multi-objective case using the Pareto approach (Pareto 1896). For each generation, we transform the population P by crossing the non-dominated solutions and mutating the dominated solutions. Then, we concatenate the current population P and the new individuals created by crossover and mutation (see Alberto and Mateo 2011). The new population is then obtained by keeping all non-dominated solutions. In case the number of non-dominated solutions is less than the population size, we complete the remaining population by the best individuals according to three fairly studied criteria: the one-third of the remaining population by the best individuals according to the makespan criterion and the one-third of the remaining population by the best individuals according to the total tardiness criterion. The best individuals according to the total completion time criterion will complement the rest of the population. In the last generation, only non-dominated solutions are kept.

4.4 NSGA-II algorithm

The NSGA-II algorithm is based on the following principles (Deb et al. 2002):

- With each generation g, merging the population of parents P_g of size s with the population of children E_g of the same size to build a new population $R_g = P_g \cup E_g$ of size $2 * s$.
- Sort the R_g results population according to the non-dominance criterion. This makes it possible to distribute R_g in several fronts $(F_1, F_2, ...)$. The first fronts contain the best individuals.
- Building the new parent population P_{g+1} by adding the F_i fronts while the size of P_{g+1} does not exceed s. In the case where the size of the new population is less than s, the crowding method is applied.

The calculation of the crowding distance of an individual is based on the following principles:

- Repeat these steps for all objectives.
- Sort the solutions of an objective in ascending order.
- Assign infinite distance for the individuals having extreme values (the first and last according to the sorts).
- For all other individuals, calculate the normalized difference of the two adjacent solutions. Add the value and calculate the distance of the current individual.

After calculating the crowding distance of the i^{th} front F_i from R_g, the list of F_i solutions must be sorted in a descending order. The best solution is selected using the crowded comparison-operator ($\prec_n$); between two different rank solutions, we choose the one with the smallest rank, if they have the same rank we choose the solution that has the greatest crowding distance.

4.5 Constructive algorithm based on lower bounds

We exploit and combine the ideas of the proposed lower bounds for each criterion (makespan, total completion time and total tardiness) to build a new Constructive Algorithm Based on Lower Bounds (CABLB). The main ideas of this algorithm is to create a feasible schedule using the lower bounds on each processors for each criterion. To diversify the selected solutions we transform the set of solutions obtained

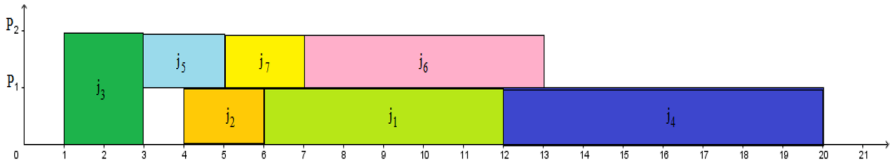

Fig. 2 Feasible schedule using the lower bounds LBTC on P_1

by crossing the non-dominated solutions and mutating the dominated solutions. At the end, only the non-dominated solutions are kept.

Example Let us consider the instance in Table 1. Now, we apply the lower bound LBTC to this instance. We obtain two sub-problems on each processor. On the first processor P_1, we have to perform the following tasks $\{j_1, j_2, j_3, j_4\}$. According to the SRPT rule (see Fig. 1), we obtain the following tasks $\{j_3, j_2, j_1, j_4\}$ to create a feasible schedule with a lower bound on the completion time on P_1.

The next step is to insert on the second processor P_2 the mono-processor tasks $\{j_5, j_6, j_7\}$ which only request the second processor P_2 according to their order by using the SRPT rule. We have to perform the following tasks $\{j_5, j_7, j_6\}$. Figure 2 represents a feasible schedule using the lower bounds LBTC on P_1 applied to the instance in Table 1.

Similarly, we create a feasible schedule using the lower bound LBTC on P_2. Then, we insert on the first processor P_1 the mono-processor tasks which only request the first processor P_1 according to their order by using the SRPT rule.

To generate the set of solutions, we applied this principle to all the proposed lower bounds (LBC, LBTC and LBTT) on each processor.

The CALB algorithm is based on the following steps:

- Step 1 Create a feasible schedule s_1 using the lower bounds LBTC on P_1 and the mono-processor tasks which only request the second processor P_2 according to their order by the SRPT rule.
- Step 2 Create a feasible schedule s_2 using the lower bounds LBTC on P_2 and the mono-processor tasks which only request the first processor P_1 according to their order by the SRPT rule.
- Step 3 Create a feasible schedule s_3 using the lower bounds LBTT on P_1 and the mono-processor tasks which only request the second processor P_2 according to their order by the SRPT rule.
- Step 4 Create a feasible schedule s_4 using the lower bounds LBTT on P_2 and the mono-processor tasks which only request the first processor P_1 according to their order by the SRPT rule.
- Step 5 Create a feasible schedule s_5 using the lower bounds LBC on P_1 and the mono-processor tasks which only request the second processor P_2 according to their order by the release dates.

Table 3 Problem types

Number tasks	*Type*1	*Type*2	*Type*3	*Type*4	*Type*5
$n_1 =$	n	n	n	n	$[n/2]$
$n_2 =$	$[n/2]$	n	$[n/2]$	n	$[n/2]$
$n_{12} =$	$[n/2]$	$[n/2]$	n	n	n

- Step 6 Create a feasible schedule s_6 using the lower bounds LBC on P_2 and the mono-processor tasks which only request the first processor P_1 according to their order by the release dates.
- Step 7 Generate from the set of solutions $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ a new set of solutions S' by crossing the non-dominated solutions and mutating the dominated solutions.
- Step 8 Select from $S \cup S'$ the non-dominated solutions.

5 Numerical results

In this section, we present some experimental results obtained on randomly generated instances. Then, we analyze these results and we provide some conclusions.

We implemented our genetic algorithm using a *DEV C++* compiler on an Intel *Core*^{TM} i3 4005U CPU 1.7 GHz, 1.7 GHz and 4 GB of RAM.

We randomly generated instances by taking into account five types of problems illustrated in Table 3 presented by Manaa and Chu (2010). The parameter n is an integer ($n \in \{10, 20\}$), and [x] corresponds to the integer part of x. The variables n_1, n_2 and n_{12}, respectively, represent the number of $P_1 - tasks$, $P_2 - tasks$ and $P_{12} - tasks$.

For these five types of problems (Manaa and Chu 2010), considered the distribution of the three types of tasks and the number of tasks on each processor (load on the processor).

For *Type*4, the distribution of tasks is balanced ($n_1 = n_2 = n_{12} = n$) and the distribution of the load on each processor ($P1 : n_1 + n_{12} = 2n$ and $P2 : n_2 + n_{12} = 2n$) is, therefore, balanced.

For *Type*1, the number of tasks n_1 exceeds that of the two other types ($n_1 > n_2$ and $n_1 > n_{12}$), while the processor $P1$ is more loaded than $P2$. For *Type*5, the number of tasks P_{12}, which requires the use of the two processors, exceeds that of tasks of the other two types ($n_{12} > n_1$ and $n_1 > n_2$). But, the distribution of load on the processors is balanced.

For *Type*2, the load on the processors is balanced, which is not the case for *Type*3. The processing times are randomly generated from the set $\{0, .., 50\}$.

The values r_j are randomly generated from the set $\{0..L\}$, with L equal to the integer part of: $\alpha * (s_1 + s_2 + s_{12})$, where $\alpha \in \{0.5, 1, 1.5\}$ and s_1, s_2 and s_{12} are, respectively, the totals of the processing time of $P_1 - tasks$, $P_2 - tasks$ and $P_{12} - tasks$.

The due dates d_j are randomly generated from the set $\{r_j + p_j, .., r_j + p_j + L\}$.

We consider that the group of instances represents the set of instances having the same parameters n, α and *Type*.

In the next subsection, we will detail the different parameters used to develop the proposed algorithms and we will present the proposed metrics used to measure the quality of our algorithms.

5.1 Parameters and metrics

It was shown that the effectiveness of a genetic algorithm depends on the size of the population, the maximum number of generations and the crossover and mutation rate.

The size of the population should vary between N and $3N$ (where N is the problem size). The maximum number of generations should represent a compromise of the quality of the solutions and the computation time. To obtain new structures in the population, the crossover rate should generally vary between 60 and 100% and the mutation rate should be between 0.1 and 5%.

In our GA (Aggregative approach and NSGA-II algorithm), we applied these parameters. In Pareto approach, we transform the current population by crossing the non-dominated solutions and by muting the dominated solutions. In this case, the number of non-dominated solutions represents the crossing rate and the number of dominated solutions is equal to the mutation rate.

We fixed the number of generations to $2Nb$ for each population where Nb is the number of tasks to be processed. The size of the population is then Nb. For the experimental results, ten instances of each group are generated and the average values are provided. Some preliminary tests have motivated our choices.

We used two metrics to measure the quality of the Pareto front: the hypervolume indicator (HV) and the number of non-dominated solutions in Pareto front (ND). The hypervolume is one of the most popular metrics for multi-objective optimisation problems (Bradstreet 2011; Lopez-Ibanez and Stutzle 2014). For that, we will use this indicator to measure and compare the performance of the aggregative, Pareto, NSGA-II and CABLB algorithms proposed to solve the studied problem.

To calculate the hypervolume of a set of non-dominated points, we used the program implemented by Fonseca et al. (2018). The hypervolume measure uses a reference point (the worst value in each criterion). In this paper, we use the initial solutions calculated by the proposed methods to determine the reference points.

To compare the performance of the proposed algorithms, we used the HV ratio (denoted by HV_r). The following formula compute HV_r the distance between the solutions and the lower bounds:

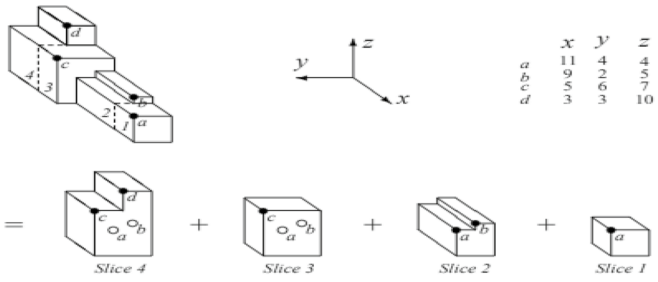

Fig. 3 One step of calculation of the hypervolume using HSO algorithm (reproduced from While et al. (2006))

Table 4 Computation time for n=10 (in second)

$n = 10$	*Type*1	*Type*2	*Type*3	*Type*4	*Type*5
$\alpha = 0.5$	0.086	0.120	0.188	0.260	0.111
$\alpha = 1$	0.135	0.111	0.141	0.134	0.157
$\alpha = 1.5$	0.098	0.128	0.122	0.130	0.115
Note: Aggregative GA results					
$\alpha = 0.5$	0.040	0.055	0.126	0.100	0.058
$\alpha = 1$	0.060	0.059	0.078	0.81	0.091
$\alpha = 1.5$	0.048	0.073	0.099	0.088	0.084
Note: Pareto GA results					
$\alpha = 0.5$	0.072	0.121	0.188	0.260	0.112
$\alpha = 1$	0.128	0.112	0.151	0.144	0.158
$\alpha = 1.5$	0.104	0.129	0.122	0.130	0.125
Note: NSGA-II results					
$\alpha = 0.5$	0.038	0.068	0.092	0.112	0.056
$\alpha = 1$	0.046	0.066	0.087	0.108	0.052
$\alpha = 1.5$	0.042	0.072	0.069	0.068	0.050

CABLB results

$$HV_r = \left(1 - \frac{HV_{LB} - HV_{Algorithm}}{HV_{LB}}\right) \tag{20}$$

Where, HV_{LB} is the volume of the space covered by the point represent the lower bound and the reference point (initial solutions).

Let a(11, 4, 4), b(9, 2, 5), c(5, 6, 7) and d(3. 3, 10) be a set of non-dominated points presented in an orthonormal coordinate system. The coordinates (x, y, z) correspond to criterion 1, criterion 2, and criterion 3, respectively. The hypervolume of the set is the volume of the space covered by points a-b-c-d.

To calculate the hypervolume, (Fonseca et al. 2018) implemented they program using HSO algorithm (Hypervolume by Slicing Objectives) proposed by While et al. (2006).

Figure 3 presents one step in HSO algorithm, including the slicing of the hypervolume, the allocation of points to each slice, and the elimination of newly dominated points. As an illustration of the calculation of the hypervolume using HSO algorithm, the reader could consult (While et al. 2006).

The remainder of this section is organized as follows. In Sect. 5.2, we will present the numerical results in terms of average computation of time. Section 5.3 compares the results of the four proposed approaches.

5.2 Computation time

Tables 4 summarizes the numerical results in terms of average computation of time (seconds). These results show the importance of distinguishing not only the total

Table 5 Computation time for n=20 (in second)

$n = 20$	*Type*1	*Type*2	*Type*3	*Type*4	*Type*5
$\alpha = 0.5$	0.152	0.170	0.237	0.225	0.168
$\alpha = 1$	0.186	0.180	0.300	0.278	0.192
$\alpha = 1.5$	0.139	0.217	0.227	0.285	0.204
Note: Aggregative GA results					
$\alpha = 0.5$	0.085	0.125	0.105	0.088	0.086
$\alpha = 1$	0.071	0.120	0.228	0.287	0.240
$\alpha = 1.5$	0.124	0.161	0.223	0.250	0.168
Note: Pareto GA results					
$\alpha = 0.5$	0.152	0.170	0.241	0.218	0.172
$\alpha = 1$	0.186	0.178	0.281	0.289	0.186
$\alpha = 1.5$	0.140	0.121	0.231	0.300	0.192
Note: NSGA-II results					
$n = 10$	*Type*1	*Type*2	*Type*3	*Type*4	*Type*5
$\alpha = 0.5$	0.082	0.086	0.122	0.142	0.088
$\alpha = 1$	0.102	0.116	0.128	0.164	0.102
$\alpha = 1.5$	0.096	0.104	0.116	0.158	0.110

Note: CABLB results

number of tasks and the length of the release dates interval, but also the different types of problems. The results also show that instances corresponding to the problem of *Type*4 (with the largest number of tasks compared to other types and the tightest distribution of the release dates with $\alpha = 0.5$) are the most difficult to solve.

Table 6 Quality of Aggregative GA for $n = 10$

(*Type*, α)	*C*/*LBC*	*TT*/*LBTT*	*TC*/*LBTC*	*ND*	$HV_r(\%)$
(1, 0.5)	**1.111**	**1.277**	**1.410**	2.5	**43.479**
(1, 1)	**1.124**	4.542	1.212	2.8	**45.390**
(1, 1.5)	**1.028**	8.200	1.160	2.8	**78.070**
(2, 0.5)	**1.120**	3.163	**1.505**	2.3	**44.134**
(2, 1)	**1.111**	5.054	1.360	3.2	46.209
(2, 1.5)	1.089	35.000	1.350	2.9	44.287
(3, 0.5)	1.129	2.155	**1.718**	2.3	30.175
(3, 1)	**1.128**	8.323	1.389	2.4	**45.802**
(3, 1.5)	**1.103**	20.222	1.341	2.6	**39.948**
(4, 0.5)	1.192	1.943	1.781	2.5	31.052
(4, 1)	1.201	12.659	**1.466**	2.6	35.217
(4, 1.5)	1.151	15.938	1.325	2.5	24.329
(5, 0.5)	**1.121**	1.172	1.602	2.5	**40.113**
(5, 1)	**1.087**	3.214	1.274	2.3	**64.183**
(5, 1.5)	**1.031**	13.889	1.198	2.8	**78.984**

Table 7 Quality of Pareto GA for $n = 10$

(*Type*, α)	*C*/*LBC*	*TT*/*LBTT*	*TC*/*LBTC*	*ND*	$HV_r(\%)$
(1, 0.5)	1.215	1.614	1.499	2.3	14.913
(1, 1)	1.194	**2.958**	**1.162**	2.9	26.078
(1, 1.5)	1.046	7.900	1.151	2.1	66.143
(2, 0.5)	1.326	3.050	1.582	2.2	43.716
(2, 1)	1.243	**3.000**	**1.298**	**3.3**	12.818
(2, 1.5)	1.050	15.222	1.181	**3.2**	65.201
(3, 0.5)	1.257	2.521	1.808	2.9	**39.814**
(3, 1)	1.217	**4.677**	1.328	**3.3**	22.822
(3, 1.5)	1.133	**6.222**	**1.258**	3.2	29.033
(4, 0.5)	1.244	**1.385**	1.652	2.9	20.596
(4, 1)	**1.141**	**5.205**	1.335	3.7	**50.022**
(4, 1.5)	1.137	**11.469**	**1.297**	**4.1**	28.614
(5, 0.5)	1.289	1.167	1.629	3.1	24.805
(5, 1)	1.120	**2.786**	**1.213**	**4.1**	52.979
(5, 1.5)	1.121	9.000	**1.118**	2.9	35.172

Table 8 Quality of NSGA-II for $n = 10$

(*Type*, α)	*C*/*LBC*	*TT*/*LBTT*	*TC*/*LBTC*	*ND*	$HV_r(\%)$
(1, 0.5)	1.181	1.422	1.449	**4.3**	22.261
(1, 1)	1.209	4.929	1.188	**4.7**	22.822
(1, 1.5)	1.077	7.100	**1.143**	**2.9**	47.854
(2, 0.5)	1.154	3.025	1.530	**2.4**	33.307
(2, 1)	1.202	3.054	1.304	2.6	20.354
(2, 1.5)	1.058	18.222	**1.204**	1.9	60.583
(3, 0.5)	1.157	2.294	1.751	**3.5**	21.338
(3, 1)	1.186	4.935	**1.267**	3.1	29.630
(3, 1.5)	1.156	11.167	1.271	**3.5**	21.746
(4, 0.5)	1.225	1.930	1.700	**3.7**	24.070
(4, 1)	1.208	7.114	**1.330**	2.2	33.697
(4, 1.5)	1.177	13.813	1.335	2.6	17.627
(5, 0.5)	1.156	**1.036**	**1.567**	**5.1**	29.078
(5, 1)	1.212	2.893	1.281	3.6	29.292
(5, 1.5)	1.148	**6.444**	1.162	**3.6**	26.050

For the aggregative and NSGA-II methods, our genetic algorithm requires an average of computation time equal to 0.260 s for the type of problem *Type*4 (with $\alpha = 0.5$). For the Pareto method, the average of computation time is equal to 0.100 s. The problem of *Type*1 remains the easiest to solve. The numerical results also reveal that the aggregative GA, NSGA-II and Pareto GA require an average of computation time more than the CABLB algorithm.

From Table 5, our genetic algorithm with NSGA-II approach requires an average computation time equal to 0, 300 s for the type of problem *Type*4 (with $\alpha = 1.5$). In

Table 9 Quality of CABLB for n=10

(*Type*, α)	*C*/*LBC*	*TT*/*LBTT*	*TC*/*LBTC*	*ND*	$HV_r(\%)$
(1, 0.5)	1.264	1.687	1.419	1.8	7.636
(1, 1)	1.254	4.917	1.193	2.2	14.701
(1, 1.5)	1.144	**6.800**	1.165	2.1	20.965
(2, 0.5)	1.154	**2.925**	1.508	2.0	33.703
(2, 1)	1.243	3.270	1.299	2.8	12.818
(2, 1.5)	**1.047**	**14.222**	1.216	2.6	**66.792**
(3, 0.5)	**1.124**	**2.000**	1.865	2.8	31.851
(3, 1)	1.233	5.097	1.337	2.6	19.850
(3, 1.5)	**1.103**	6.778	1.286	3.0	**39.948**
(4, 0.5)	**1.178**	1.411	1.668	2.1	**34.416**
(4, 1)	1.265	6.000	1.351	2.7	23.080
(4, 1.5)	**1.133**	12.094	1.302	3.0	**30.146**
(5, 0.5)	1.225	1.176	1.636	2.4	13.461
(5, 1)	1.255	2.821	1.295	3.0	21.109
(5, 1.5)	1.169	8.667	1.180	2.6	20.354

Manaa and Chu (2010), the branch-and-bound algorithm to minimize the makespan criterion, needs in average more than 574 s to find the optimal solution. The problem of *Type*1 (having the smallest number of tasks compared to others) requires less computation time compared to other problems. This can be justified by the fact that the processors are loaded with less than the number of bi-processor tasks compared to other cases.

5.3 Solution quality

The following fields (*C*; *TT*; *TC*) denote respectively the makespan, the total tardiness and the total completion time. (*LBC*; *LBTT*; *LBTC*) denote, respectively, the lower bounds of makespan, total tardiness and total completion time.

To compare the three objectives of our Aggregative, Pareto, NSGA and CABLB algorithms, we use the average quality of the three objectives: *C*/*LBC*, *TT*/*LBTT* and *TC*/*LBTC*. Furthermore, the number of non-dominated solutions and HV ratio will be used to measure the performance of the four proposed algorithms.

Tables 6, 7, 8 and 9 present the results of the aggregative, Pareto, NSGA-II and CABLB algorithms for $n = 10$. Column 1 indicates the types of problems (*Type*, α) considered in the distribution of the three types of tasks and the number of tasks on each processor. In columns 2, 3 and 4, (*C*/*LBC*, *TT*/*LBTT*, *TC*/*LBTC*) indicates the average quality of the makespan, the total tardiness and the total completion time, respectively, from instances randomly generated. Column 5 presents the average number of non-dominated solutions for each problem. Finally, in column 6, we present the results of the HV ratios.

The results for the four algorithms listed in Tables Tables 6, 7, 8 and 9 show that the aggregative GA is more effective on the makespan criterion for the problems of *Type*1, *Type*2 with($\alpha = 0.5$, $\alpha = 1$), *Type*3 with($\alpha = 1$, $\alpha = 1.5$) and *Type*5. For the

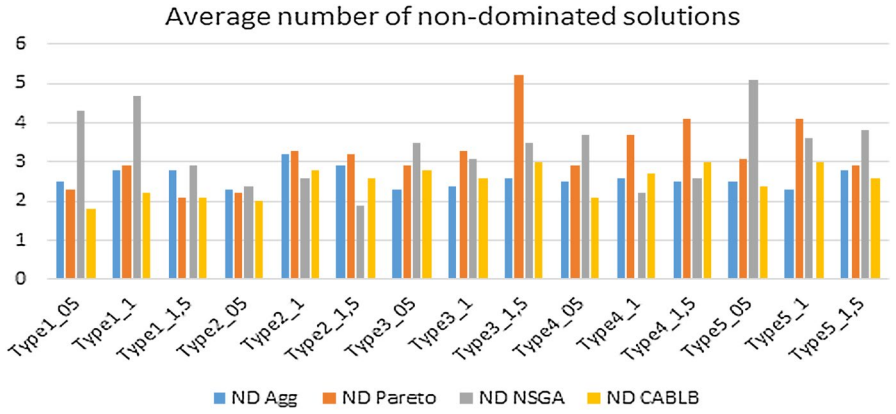

Fig. 4 Average number of non-dominated solutions for n =1

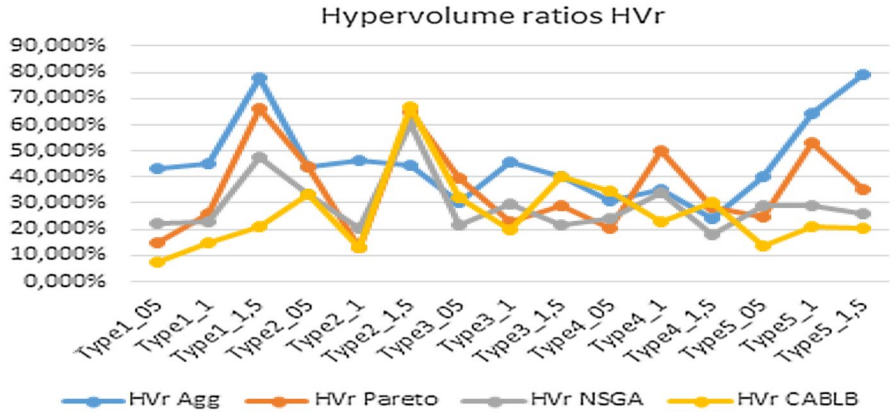

Fig. 5 Hypervolume ratios for $n = 10$

problems of *Type*2 with ($\alpha = 1.5$), *Type*3 with ($\alpha = 0.5$, $\alpha = 1.5$) and *Type*4 with ($\alpha = 0.5$, $\alpha = 1.5$) CABLB algorithm is more efficient. For the problems of *Type*4 with ($\alpha = 1$) Pareto is more efficient.

The quality of the solutions found by the NSGA-II approach is good on total tardiness criterion for the problem of *Type*5 with ($\alpha = 0.5$, $\alpha = 1.5$). For the problem of *Type*1 with ($\alpha = 1$), *Type*2 with ($\alpha = 1$), *Type*3 with ($\alpha = 1$, $\alpha = 1.5$), *Type*4 and *Type*5 with ($\alpha = 1$) Pareto is more efficient. The results of aggregative GA are very bad for the problem of *Type*2 with ($\alpha = 1.5$) and more efficient for the problem of *Type*1 with ($\alpha = 0.5$). For the problems of *Type*1 with ($\alpha = 1.5$), *Type*2 with ($\alpha = 0.5$, $\alpha = 1.5$) and *Type*3 with ($\alpha = 0.5$) CABLB algorithm is more efficient on total tardiness criterion.

For the total completion time criterion, the results with NSGA-II algorithm is more efficient. The results with the aggregative GA and Pareto GA are close to the lower bounds in some cases and quite far from these lower bounds for the other

Table 10 Quality of aggregative GA for $n = 20$

$(Type, \alpha)$	*C/LBC*	*TT/LBTT*	*TC/LBTC*	*ND*	$HV_r(\%)$
(1, 0.5)	1.213	4.339	1.923	2.3	21.058
(1, 1)	1.205	15.904	1.512	1.9	31.696
(1, 1.5)	**1.060**	58.857	1.412	1.7	**61.917**
(2, 0.5)	1.298	3.328	1.984	2.5	19.100
(2, 1)	1.210	15.962	1.630	2.2	21.514
(2, 1.5)	1.108	99.889	1.579	2.0	38.618
(3, 0.5)	1.263	4.803	2.204	1.4	16.517
(3, 1)	1.302	11.643	1.703	1.8	19.300
(3, 1.5)	1.157	86.722	1.593	2.4	28.814
(4, 0.5)	1.357	4.345	2.198	**2.7**	12.109
(4, 1)	1.303	20.399	1.823	2.6	12.395
(4, 1.5)	**1.134**	45.449	1.629	2.2	**35.425**
(5, 0.5)	1.299	2.827	2.057	**2.4**	16.703
(5, 1)	1.232	9.337	1.592	1.0	27.990
(5, 1.5)	1.132	26.362	1.519	2.3	37.857

Table 11 Quality of Pareto GA for $n = 20$

$(Type, \alpha)$	*C/LBC*	*TT/LBTT*	*TC/LBTC*	*ND*	$HV_r(\%)$
(1, 0.5)	1.249	3.751	1.868	**3.2**	14.496
(1, 1)	**1.156**	**7.137**	**1.372**	2.3	**43.479**
(1, 1.5)	1.062	**40.429**	**1.356**	**5.6**	61.002
(2, 0.5)	**1.292**	**2.331**	**1.890**	**5.3**	**20.002**
(2, 1)	1.189	**12.278**	**1.512**	**5.2**	26.078
(2, 1.5)	**1.083**	**68.815**	**1.507**	**4.4**	**49.480**
(3, 0.5)	**1.188**	**3.328**	**2.124**	**4.1**	**31.028**
(3, 1)	**1.198**	**6.889**	1.628	**4.1**	**37.696**
(3, 1.5)	1.188	**47.417**	1.538	**5.9**	20.810
(4, 0.5)	1.352	3.719	**2.111**	1.7	12.632
(4, 1)	1.293	19.214	1.928	**3.1**	13.698
(4, 1.5)	1.165	43.494	1.572	**6.1**	26.176
(5, 0.5)	**1.296**	**1.994**	**1.912**	2.2	**17.147**
(5, 1)	**1.111**	**4.251**	**1.398**	**10.2**	**58.132**
(5, 1.5)	1.141	**11.224**	**1.351**	**6.6**	35.141

cases. The results found by the CABLB algorithm are quite far from these lower bounds on total completion time criterion.

The graphical representation of the front size described in Fig. 4 shows that the space of solutions found by NSGA-II and Pareto technique are the most diverse in many cases containing a significant number of non-dominated solutions (between 2 and 5 solutions). This is justified by the fact that this approach ensures elitism by archiving non-dominated solutions in the evolution from one generation to another.

Table 12 Quality of NSGA-II for $n = 20$

$(Type, \alpha)$	C/LBC	$TT/LBTT$	$TC/LBTC$	ND	$HV_r(\%)$
(1, 0.5)	**1.179**	**3.257**	**1.618**	2.0	**28.519**
(1, 1)	1.196	12.904	1.508	**2.5**	33.858
(1, 1.5)	1.089	44.762	1.400	1.8	47.563
(2, 0.5)	1.351	2.889	1.907	2.7	12.500
(2, 1)	**1.180**	14.128	1.571	2.8	**28.071**
(2, 1.5)	1.122	76.519	1.514	3.0	33.336
(3, 0.5)	1.242	4.342	2.162	1.8	19.925
(3, 1)	1.346	8.609	**1.597**	2.0	13.705
(3, 1.5)	**1.146**	66.500	**1.494**	1.6	**32.168**
(4, 0.5)	1.342	**1.943**	2.144	2.4	13.722
(4, 1)	**1.175**	**15.679**	**1.515**	2.6	**35.770**
(4, 1.5)	1.192	**40.831**	**1.571**	3.2	19.474
(5, 0.5)	1.379	2.517	2.043	2.0	7.948
(5, 1)	1.240	6.611	1.591	2.5	26.567
(5, 1.5)	**1.125**	13.500	1.399	2.1	**40.128**

Table 13 Quality of CABLB for $n = 20$

$(Type, \alpha)$	C/LBC	$TT/LBTT$	$TC/LBTC$	ND	$HV_r(\%)$
(1, 0.5)	1.219	4.374	1.641	2.1	19.742
(1, 1)	1.196	16.219	1.514	2.0	33.856
(1, 1.5)	1.095	47.524	1.419	1.8	45.287
(2, 0.5)	1.310	3.061	1.903	2.2	17.377
(2, 1)	1.243	14.496	1.616	2.3	15.374
(2, 1.5)	1.090	77.815	1.577	2.1	46.282
(3, 0.5)	1.268	4.509	2.264	1.9	15.731
(3, 1)	1.352	10.821	1.657	1.8	13.039
(3, 1.5)	1.168	70.722	1.527	2.0	25.702
(4, 0.5)	**1.295**	4.255	2.200	2.3	**19.760**
(4, 1)	1.236	16.583	1.607	2.0	22.526
(4, 1.5)	1.182	46.315	1.659	2.1	21.940
(5, 0.5)	1.338	2.761	2.059	2.3	11.964
(5, 1)	1.217	8.240	1.576	1.9	30.987
(5, 1.5)	1.175	17.052	1.500	2.1	25.575

The solution space remains the least diversified with the aggregative GA and CABLB algorithm, but they are most effective for the makespan criterion.

Figure 5 presents the results of the HV ratios for the four proposed algorithms. The results for the three criteria show that the aggregative selection technique is more effective for problems of $Type1$, $Type2$ with ($\alpha = 1$), $Type3$ with ($\alpha = 1$) and $Type5$. For the problems of $Type3$ with ($\alpha = 0.5$, $\alpha = 1.5$) and $Type4$ with ($\alpha = 1.5$), the NSGA-II algorithm is slightly less efficient. For the CABLB algorithm, the HV ratio is very bad and strongly decreases with the problem of $Type1$,

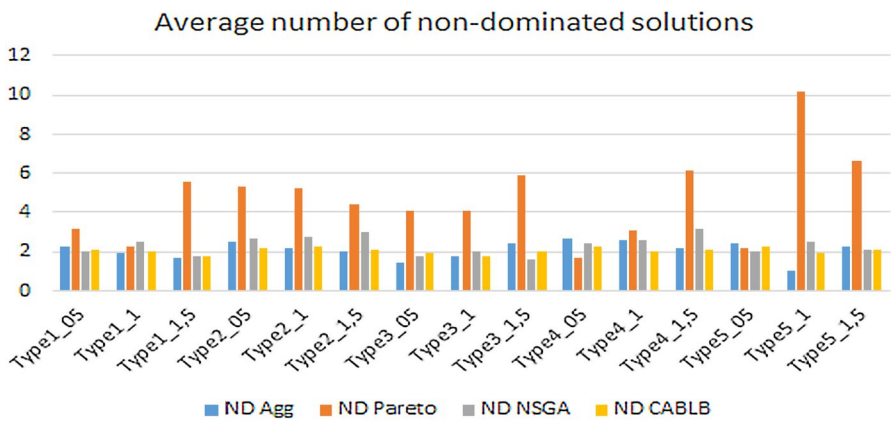

Fig. 6 Average number of non-dominated solutions for $n = 20$

*Type*2 with ($\alpha = 1$, $\alpha = 1.5$) and *Type*5. It is more effective for problems of *Type*2 with ($\alpha = 1.5$).

For the problems of *Type*4 with ($\alpha = 0.5$), the Pareto GA less efficient. It is more effective for problems of *Type*4 with ($\alpha = 1$).

Tables 10, 11, 12 and 13 present the results of the four proposed algorithms for $n = 20$.

The results for the makespan criterion described in these tables show that the aggregative GA is more effective for the problems of *Type*1 with ($\alpha = 1.5$) and *Type*4 with ($\alpha = 1.5$). For the problems of *Type*1 with ($\alpha = 1$), *Type*2 with ($\alpha = 0.5$, $\alpha = 1.5$), *Type*3 with ($\alpha = 0.5$, $\alpha = 1$) and *Type*5 with ($\alpha = 0.5$, $\alpha = 1$) Pareto GA is more effective. For the problems of *Type*1 with ($\alpha = 0.5$), *Type*2 with ($\alpha = 1$), *Type*3 with ($\alpha = 1.5$), *Type*4 with ($\alpha = 1$) and *Type*5 with ($\alpha = 1.5$) NSGA-II is more effective. For other cases, the CABLB is more effective. For

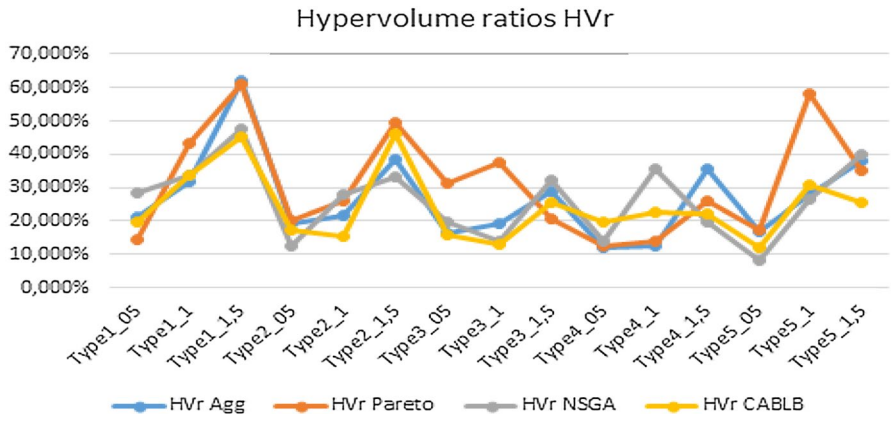

Fig. 7 Hypervolume ratios for $n = 20$

the five types of problems studied with the makespan criterion, the NSGA-II and Pareto are identical.

The results for the total tardiness criterion described in Tables 10 to 13 show that the Pareto approach is more effective for the problems of *Type*1 with ($\alpha = 1$, $\alpha = 1.5$), *Type*2, *Type*3 and *Type*5. For the problem of *Type*1 with ($\alpha = 0.5$) and *Type*4 NSGA-II is the most effective. The aggregative GA and CALBLB algorithm are less effective on the total tardiness criterion.

The numerical results of the total completion time criterion show that the Pareto technique is more effective in many cases (*Type*1 with ($\alpha = 1$, $\alpha = 1.5$), *Type*2, *Type*3 with ($\alpha = 0.5$), *Type*4 with ($\alpha = 0.5$) and *Type*5). For other cases, NSGA-II is more effective. By looking at these numerical values, we conclude that the four approaches are almost identical and the averages values found are very close.

Figure 6 summarizes average number of non-dominated solutions for $n = 20$. The space of solutions found by Pareto technique are the most diverse containing a significant number of non-dominated solutions (between 2 and 10 solutions). The solution space remains the least diversified with the aggregative GA and CABLB algorithm.

Figure 7 presents the results of the HV ratios for the four proposed algorithms (for $n = 20$). The results for the three criteria show that the Pareto technique is more effective for problems of *Type*1 with ($\alpha = 1$), *Type*2 with ($\alpha = 0.5$, $\alpha = 1.5$), *Type*3 with ($\alpha = 0.5$, $\alpha = 1$) and *Type*5 with ($\alpha = 0.5$, $\alpha = 1$). For the problems of *Type*1 with ($\alpha = 1.5$) and *Type*4 with ($\alpha = 1.5$), the aggregative GA is more efficient.

For the NSGA method, the HV ratio is very bad and strongly decreases with the problem of *Type*2 with ($\alpha = 0.5$), *Type*4 with ($\alpha = 0.5$) and *Type*5 with ($\alpha = 0.5$). For the problems of *Type*1 with ($\alpha = 0.5$), *Type*2 with ($\alpha = 1$), *Type*3 with ($\alpha = 1.5$), *Type*4 with ($\alpha = 1$) and *Type*5 with ($\alpha = 1.5$) the NSGA-II method is more efficient.

For the CABLB algorithm, the HV ratio is very bad and strongly decreases with the problem of *Type*2 with ($\alpha = 1$) and *Type*3 with ($\alpha = 0.5$, $\alpha = 1$). For the problems of *Type*4 with ($\alpha = 0.5$) the CABLB algorithm is more efficient.

The results found by four approaches listed in the Table 6 to 13 are close to the lower bounds for the makespan criterion. In many cases, the results for the total completion time are close to the lower bounds for NSGA-II and Pareto approach. For total tardiness, the results are quite far from lower bounds with the four approaches studied.

6 Conclusions and perspectives

We studied a multi-objective scheduling problem on two dedicated processors to optimize three criteria; the makespan, the total tardiness and the total completion time. In this study, we exploited the lower bound constructed for each criterion, the hypervolume indicator (HV) and the number of solutions in the optimal front (ND) to assess the quality of the solutions found by aggregative GA, Pareto GA, NSGA-II and CABLB algorithm proposed for solving the multi-objective problem. To

generate the weight for the aggregative GA, we used the method of Uniform Design (proposed by Leung and Wang (2000)) to choose a variety of solutions uniformly dispersed.

The results of the studied problems are encouraging and promising for the makespan and the total completion time criteria. Each studied technique (aggregative, Pareto, NSGA-II or CABLB) has an advantage compared to the others according one criterion. In some cases, Pareto techniques and NSGA-II are almost identical. Therefore, it is interesting to study other extensions of these problems in a future work, like the study of scheduling problem on parallel processors. For example minimizing the makespan for the problem $P2|size_j, r_j|C_{\max}$ where $size_j$ is the number of processors required by the task j. This problem was proved NP-hard in the strong sense in Blazewicz et al. (2002). Therefore, it is interesting to test the proposed methods for scheduling these problems on parallel processors (Vallada and Ruiz 2012; Venkata et al. 2018).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

Afrati F, Milis I (2006) Designing ptass for min-sum scheduling problems. Discr Appl Math 154(4):622–639

Afrati F, Bampis E, Chekuri C, Karger D, Kenyon C, Khanna S, Milis I, Queyranne M, Sartinutella M, Stein C, Sviridenko M (1999) Approximation schemes for minimizing average weighted completion time with release dates. In: IEEE symposium on foundations of computer science, pp 32–44

Alberto I, Mateo P (2011) A crossover operator that uses pareto optimality in its definition. Top 19(1):67–92

Alhadi G, Kacem I, Laroche P, Osman I (2020) Approximation algorithms for minimizing the maximum lateness and makespan on parallel machines. Ann Oper Res 285:369–395

Berrichi A, Amodeo L, Yalaoui F, Chatelet E, Mezghiche M (2007) Biobjective optimization algorithms for joint production and maintenance scheduling: application to the parallel machine problem. J Intell Manuf 20(4):389–400

Bianco L, Blazewicz J, Dell'Olmo P, Drozdowski M (1997) Preemptive multiprocessor task scheduling with release times and time windows. Ann Oper Res 70(1):43–55

Blazewicz J, Dell'Olmo P, Drozdowski M (2002) Scheduling multiprocessor tasks on two parallel processors. RAIRO-Oper Res 36(1):37–51

Blazewicz J, Ecker K, Pesch E, Schmidt G, Weglarz J (2019) Handbook on scheduling. Springer, New York

Bradstreet L (2011) The hypervolume indicator for multi-objective optimisation: calculation and use. PhD thesis, University of Western Australia

Chu C (1992) A branch and bound algorithm to minimize the total of tardness with different release date. Naval Res Log 39(2):256–283

Coffman E, Garey M, Johnson D, LaPaugh AS (1985) Scheduling file transfers. SIAM J Comput 14(4):743–780

Craig G, Kime CR, Saluja K (1988) Test scheduling and control for vlsi built-in self-test. IEEE Trans Comput 37(9):1099–1109

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga ii. IEEE Trans Evol Comput 6(2):182–197

Drozdowski M (1996) Scheduling multiprocessor tasks an overview. Eur J Oper Res 94(2):215–230

Emmons H (1969) Some new uniform design. Oper Res 17(4):701–715

Fang K, Li J (1994) Multi-objective genetic algorithms made easy: selection, sharing and mating restrictions. Hong Kong, Baptist Univ., Hong Kong, Tech. Rep. Math-042

Fonseca LC, Mand P, López-Ibñáez M, Guerreiro AP (2018) Computation of the hypervolume indicator. http://lopez-ibanez.eu/hypervolume. Accessed 3 Feb 2019
Holland J (1975) Adaptation in natural and artificial systems. Michigan Press, University of Michigan Press, USA
Hoogeveen J, de Velde SLV, Veltman B (1994) Complexity of scheduling multiprocessor tasks with prespecified processors allocations. Discr Appl Math 55(3):259–272
Kacem A, Dammak A (2017) A genetic algorithm to minimize the total of tardiness multiprocessing tasks on two dedicated processors. In: IEEE control, decision and information technologies, Barcelona, Spain 5-7 April. 2017:85–90
Kacem A, Dammak A (2019) Bi-objective scheduling on two dedicated processors. Eur J Ind Eng 5:681–700
Kacem I (2007) Lower bounds for tardiness minimization on a single machine with family setup times. Int J Oper Res 4(1):18–31
Karasakal E, Silav A (2016) A multi-objective genetic algorithm for a bi-objective facility location problem with partial coverage. Top 24(1):206–232
Leung Y, Wang Y (2000) Multiobjective programming using uniform design and genetic algorithm. IEEE Trans Syst Man Cybern 30(C):293–303
Li G (1997) Single machine earliness and tardiness scheduling. Eur J Oper Res 96(3):546–558
Lopez-Ibanez M, Stutzle T (2014) Automatically improving the anytime behaviour of optimisation algorithms. Eur J Oper Res 235(3):569–582
Manaa A, Chu C (2010) Scheduling multiprocessor tasks to minimise the makespan on two dedicated processors. Eur J Ind Eng 4(3):265–279
Oguz C, Ercan M (2005) A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. J Sched 8(4):323–351
Pareto V (1896) Cours économie politique. Lausane Switzerland, Switzerland
Rebai M, Kacem I, Adjallah K (2010) Earliness tardiness minimization on a single machine to schedule preventive maintenance tasks: metaheuristic and exact methods. J Intell Manuf 23(4):1207–1224
Vallada E, Ruiz R (2011) A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. Eur J Oper Res 211(3):612–622
Vallada E, Ruiz R (2012) Scheduling unrelated parallel machines with sequence dependent setup times and weighted earliness–tardiness minimization. In: Just-in-time systems, Springer, pp 67–90
Venkata P, Usha M, Viswanath K (2018) Order acceptance and scheduling in a parallel machine environment with weighted completion time. Eur J Ind Eng 12(4):535–557
While L, Hingston P, Barone L, Husband S (2006) A faster algorithm for calculating hypervolume. IEEE Trans Evol Comput 10(1):29–38