Scheduling of Jobs on Dissimilar Parallel Machine using Computational Intelligence Algorithms

Remya Kommadath and Prakash Kotecha

Abstract: The success of Computational Intelligence (CI) techniques to solve combinatorial scheduling problem critically depends on efficient modelling of the problem. In this chapter, we study the scheduling of a set of jobs with different release and due dates on a set of dissimilar parallel machine problems to minimize the processing cost. The performance of five recent CI techniques *viz.,* Artificial Bee Colony, Dynamic Neighborhood Learning based Particle Swarm Optimizer, Genetic Algorithm, Multi-Population Ensemble Differential Evolution (MPEDE) and Sanitized Teaching-Learning based Optimization is evaluated on problems reported in literature. It was observed from 750 unique trials (5 problems x 2 datasets x 5 algorithms x 15 runs) that MPEDE showed superior performance to the other four algorithms for larger problems.

Keywords: Scheduling, Artificial Bee Colony, Particle Swarm Optimization, Genetic Algorithm, Differential Evolution, Teaching-Learning based Optimization

Remya Kommadath
Department of Chemical Engineering, Indian Institute of Technology Guwahati, Assam, 781 039, India
e-mail: remya@iitg.ac.in

Prakash Kotecha
Department of Chemical Engineering, Indian Institute of Technology Guwahati, Assam, 781 039, India
e-mail: pkotecha@iitg.ac.in

1. Introduction

Scheduling involves the allocation of tasks to the available resources in order to accomplish certain objectives within specified time. In an increasingly competitive environment with several uncertainties, it becomes critical for organizations to optimally employ their limited resources for completing a given set of tasks to increase their profitability. Scheduling problems are combinatorial in nature and these have received significant attention from researchers working in the area of operations research. In a typical machine-scheduling problem, the jobs represent the tasks that are to be performed whereas the machines indicate the available resources. The aim is to sequence and schedule the processing of tasks by the set of available resources to achieve the optimal solution with respect to certain objective(s). Some of the common objectives include minimization of makespan (Lenstra, Shmoys et al. (1990)), minimization of the total processing cost (Hooker, Ottosson et al. (1999), Jain and Grossmann (2001)), and minimization of inventory cost (Dobson and Arai Yano (1994)). These problems are predominantly solved using mathematical programming techniques by explicitly posing the problem in a Mixed-Integer Linear Programming (MILP) or Mixed-Integer Non-Linear Programming (MINLP) framework. Some other techniques such as Constraint Programming (CP), heuristic methods as well as stochastic optimization techniques have also been used in relatively fewer instances. The scheduling of a set of tasks with release and due date has been considered in literature (Jain and Grossmann (2001)) and has been solved using MILP, CP, combined MILP-CP OPL model and a hybrid MILP/CP model. This work was further studied (Hooker (2005)) with the development of advanced Benders cuts for the objective of minimization of makespan for the sub-problems. This work also relied on hybridizing CP and MILP. Minimization of makespan for parallel batch processing machines with unequal ready time and arbitrary job sizes have been solved using a combination of MILP and compound algorithm along with three different heuristic techniques (Chung, Tai et al. (2009)). Several surveys are available for the scheduling problems that discuss various aspects of the problem (Pfund, Fowler et al. (2004), Allahverdi, Ng et al. (2008)).

Traditional mathematical programming techniques have been reported to be inefficient to solve large-scale scheduling problems due to an almost exponential increase in the number of binary variables and constraints. Computational Intelligence techniques possess several advantages such as (i) ability to handle nonlinearities, (ii) accommodates conflicting objectives, (iii) provides value added solutions, (iv) solves black-box optimization problems, (v) does not require information about gradients, (vi) can be tuned to accommodate problem specific operators, (viii) does not require the problem to be postulated in the conventional equality and inequality form, (ix) the number of decision variables and constraints do not exponentially increase with an increase in the problem size, and (x) can be easily integrated with parallel and GPU computing. The success of these techniques for real parameter optimization has been significantly larger than combinatorial optimization problems. Despite their advantages, it should be noted that these techniques need to be judiciously applied to combinatorial optimization problems. These might exhibit poor performance if these are used to solve such problems posed in the conventional MILP or MINLP models that use a large number of artificial variables. In many instances, their success is critically dependent on appropriate choice of decision variables (Chauhan and Kotecha (2018a), Chauhan, Sivadurgaprasad et al. (2018b)) and their bounds, modification or incorporation of specific operators (Hasda, Bhattacharjya et al. (2017)) that exploit the structure of the problem. Even though these techniques do not explicitly guarantee the global optima, these still can determine multiple reasonable schedules in a lower time as compared to the traditional methods (Mastrolilli and Gambardella (2000)). These multiple solutions can provide flexibility to the users for selecting the appropriate schedule as per their requirements.

A hybrid version of Genetic Algorithm (GA) was utilized for minimizing the makespan for scheduling parallel batch processing machines with arbitrary job sizes (Kashan, Karimi et al. (2008)). The scheduling of jobs with flexibility in assigning machines to the different operations of each job has been solved (Pezzella, Morganti et al. (2008)) using a modified GA that employs rule based assignment methods for creating initial population and also incorporates multiple strategies for reproduction and variation of the solutions. A combination of GA and Simulated Annealing (SA) has been proposed (Xia and Wu (2005)) to solve the multi-objective job shop scheduling with minimization of makespan, total work load of machines and critical machine work load as the objective functions. The parallel machine scheduling problem has also been solved with SA (Lee, Wu et al. (2006)) and a Discrete Particle Swarm Optimization along with a hybrid variant (Kashan and Karimi (2009)). A memetic algorithm based on Particle Swarm Optimization (PSO), SA, Nawaz-Enscore-Ham heuristic has also been devised to solve the flowshop scheduling problem (Liu, Wang et al. (2007)). Another hybrid version with PSO and a local search technique has also been reported (Moslehi and Mahnam (2011)) to solve the multi-objective job shop scheduling.

A large number of CI techniques(Eskandar, Sadollah et al. (2012), Mirjalili (2015), Punnathanam and Kotecha (2016)) are developed every year but some of the proposed techniques are demonstrated on trivial optimization problems and their performance is not critically evaluated on benchmark optimization problems. In many cases, there have been issues in independently implementing the algorithms due to incomplete description and conflicting results reported in literature (Črepinšek, Liu et al. (2012), Mernik, Liu et al. (2015)) including unfair comparisons. Some of these techniques have shown to be significantly better than the traditional techniques such as GA, SA and PSO on real parameter optimization problems but they have not been used or benchmarked with other algorithms for solving scheduling problems. In this work, we attempt to address this lacuna by solving the job shop scheduling problem with five recently proposed optimization techniques on various instances of a combinatorial scheduling problem. It should be noted that it has been reported that neither MILP nor CP were able to solve this problem on a standalone basis (Jain and Grossmann (2001), Kotecha, Bhushan et al. (2011)).

The chapter is organized as follows: In the succeeding section, we provide detailed description of the problem statement and the solution strategy. In Section 3, we briefly describe the five techniques which have been considered in this work. Section 4 presents the experimental settings whereas the results and its detailed analysis is provided in Section 5. We conclude by summarizing the developments in this work and suggest possible future work.

2. Problem Statement

In this problem, a set of independent orders *(I)* need to be processed on a set of dissimilar parallel machines *(M)*. A machine can process multiple orders whereas an individual order cannot be processed on multiple machines. It should be noted that a machine could process only one order at a given point of time. The processing cost and time depends on the order as well as the machine used for processing the order. The notations c_{im} and p_{im} denote the processing cost and processing time respectively of order i on machine m. Every order is associated with a release date (r_i) and due date (d_i). Pre-emption or extension of processing of orders is not permitted i.e., the processing of any order must be performed on or after the release date and has to be completed on or before the due date. The objective is to determine the minimum cost schedule such that all the orders are completed before their due dates subject to the satisfaction of all the constraints. Typically, for a MILP formulation, this problem is modelled with the inclusion of binary decision variables, say x_{im} which will be assigned a value of 1 if order i is processed on machine m and zero otherwise. Another binary variable y_{ij} is used to sequence the orders on a given machine and assumes a value of one if and only if the i^{th} and j^{th} order are processed by the same machine with the i^{th} order preceding the j^{th} order. Another set of continuous variables are employed to determine the start and end times. The number of discrete variables can exponentially increase with an increase in the number of order and machines. Such a modeling strategy may not yield successful results with respect to CI techniques. The fact that CI techniques do not require an explicit optimization formulation has been exploited to efficiently model this problem (Kotecha, Bhushan et al. (2011)) as described below.

Decision Variables: A careful analysis of the problem would indicate that there are only two decision variables associated with every order, i.e., (i) the machine used to process a particular order and (ii) the start time of the order on the machine. Hence the problem can be efficiently modelled with 2*I* decision variables which is not only independent of the number of machines but does not exponentially increase with an increase in the number of orders. For a problem with *I* orders, the decision variables can be divided into two distinct sets as in Fig. 1.

The first set indicates the machine that is assigned to each order whereas the second set indicates the starting time of each order. The first decision variable indicates the machine to which the first order is assigned, the second decision variable indicates the machine to which the second order is assigned and so on till the I^{th} decision variable. The $(I + 1)^{th}$ variable indicates the starting time of the first order, the $(I + 2)^{nd}$ decision variable indicates the starting time of the second order and so on till the starting time of the I^{th} order in the *2I* position. For example, Eq. (1) shows a potential solution for the assignment of four machines to five orders.

$$X = \left[\overbrace{2 \quad 3 \quad 1 \quad 4 \quad 3}^{Machines} \quad \underbrace{1 \quad 3 \quad 5 \quad 2 \quad 7}_{Starting\ times} \right] \tag{1}$$

In the above example, order I_1 is assigned to machine M_2, order I_2 is assigned to machine M_3, order I_3 is assigned to machine M_1 whereas orders I_4 and I_5 are assigned to machine M_4 and machine M_3 respectively. This selection of the representation of the decision variables inherently satisfies the condition that an order is not assigned to more than one machine. The second set of decision variable indicates that I_1 which is assigned to M_2 is started at time 1. Similarly, I_2 (assigned to M_3) starts at time 3 whereas I_3, I_4 and I_5 start at time 5, 2, and 7 respectively on their specified machines.

Bounds: The lower and upper bound is identical for all the decision variables in the first set with 1 being the lower bound and the maximum number of machines available being the upper bound. The lower bound for the second set of decision variables is the release date whereas the upper bound is the due date. Thus, unlike the first set of decision variables, the variables in the second set of decision variables need not have an identical lower and upper bounds.

Constraints: The following three constraints are incorporated to ensure that a feasible schedule is obtained.

(i) **Due date:** The completion time of order *i* is the sum of the starting time and the processing time of the order on its assigned machine as in Eq. (2) and it should be less than or equal to the due date of the order.

$$end_i = t_i + p(i, m_i) \quad \forall i \in I \tag{2}$$

As per the decision variable in Eq. (1), $m_1 = 2$ (since order 1 is assigned to machine 2) $m_2 = 3; m_3 = 1; m_4 = 4$ and. Thus the violation in the due date constraint can be determined as Eq. (3).

$$P_i^{due} = \begin{cases} 0 & if\ d_i \geq end_i \\ |end_i - d_i| & otherwise \end{cases} \quad \forall i \in I \tag{3}$$

(ii) **Overlapping of Orders:** If more than a single order is processed on the same machine, then the start time of the subsequent order should not be lower than the end time of the preceding order. The penalty for violation of this requirement is calculated as Eq. (4).

$$P_i^m = \begin{cases} 0 & if\ end_i < t_{i+1} \\ |end_i - t_{i+1}| & otherwise \end{cases} \quad \forall i \in \mathrm{I}_m, m \in M : t_i < t_{i+1} \tag{4}$$

It should be noted that the penalty is incurred if there is an overlapping with respect to the immediate succeeding order on that machine. In order to understand the above constraint, consider the following example, which has nine orders and three machines.

$$X = \left[\overbrace{3\ 1\ 2\ 1\ 2\ 1\ 2\ 1\ 3}^{Machines}\ \underbrace{8\ 4\ 6\ 14\ 9\ 7\ 11\ 5\ 2}_{Starting\ time} \right]$$

As per the encoding scheme explained earlier, it can be observed that I_2, I_4, I_6 and I_8 are processed on M_1 whereas I_3, I_5 and I_7 are processed on M_2 with the remaining two orders processed on M_3. Without loss of generality and for ease in understanding, let us consider that the processing time of all the nine orders on their respective machines is 3 time units. As per Eq. (4),

- I_2 begins at 4 and will end at 7. However, I_8 on M_1 begins at 5. Thus there is overlapping of orders and hence $P_1^1 = 2$.
- I_8 begins at 5 and will end at 8. However, I_6 on M_1 begins at 7. Thus there is overlapping of orders and hence $P_2^1 = 1$.
- I_6 begins at 7 and will end at 10. However, I_4 on M_1 begins at 14. Thus there is no overlapping of orders and hence $P_3^1 = 0$.

Thus the total penalty due to overlapping of orders on M_1 is 3. A similar analysis will yield that the total penalty is 1 for M_2 and 0 for M_3.

(iii) **Maximum Processing Time:** The difference between the earliest release date and the maximum due date among all the orders assigned to a particular machine provides an upper bound on the maximum time available to a machine and is determined as per Eq. (5).

$$H_m = \max_{i \in I_m}(d_i) - \min_{i \in I_m}(r_i) \quad \forall m \in M \tag{5}$$

The total amount of time required for completing all the orders on a machine can be determined as Eq. (6).

$$T_m = \sum_{i=1}^{I_m} p(i, m_i) \quad \forall m \in M \tag{6}$$

All the orders assigned to a particular machine should be completed within the available time. The amount of penalty for the violation of this constraint is calculated using Eq. (7).

$$P_m = \begin{cases} 0 & if\ H_m > T_m \\ T_m - H_m & otherwise \end{cases} \quad \forall m \in M \tag{7}$$

The total penalty for violation in the various constraints can be calculated as Eq. (8).

$$Penalty = \sum_{i=1}^{I} P_i^{due} + \sum_{m=1}^{M} \sum_{i=1}^{I_m} P_i^m + \sum_{m=1}^{M} P_m \tag{8}$$

Fitness function: The objective function is the determination of the least cost schedule for processing all the orders. The cost associated with a potential schedule is calculated using Eq. (8)

$$f = \begin{cases} \sum_{i=1}^{I} c(i,m) & if \;\; Penalty = 0 \\ \sum_{i=1}^{I} c(i,m) + Penalty + \sum_{i \in I} \max_{m \in M} \{c(i,m)\} & otherwise \end{cases} \quad (9)$$

For a feasible solution, the sum of all penalty must be equal to zero. An infeasible solution is penalized by the amount of violation of constraints as well as the maximum cost associated to process an i^{th} order. This ensures that an infeasible solution will not have a better fitness value in comparison with a feasible solution. It should be noted that the above strategy can be easily incorporated in CI techniques but cannot be incorporated in mathematical programming technique as m_i is a decision variable that is used as an index to access the corresponding data. The p-codes of the objective functions and Gantt charts can be downloaded from https://goo.gl/4eVNL5.

3 Algorithm Description

In this section, we provide a brief description of the algorithms and additional details can be obtained from the relevant cited literature. These algorithms have not only been selected in view of their popularity but also because of the availability of their exact implementations. This helps in independent verification of the results and avoids the criticism that the poor performance of an algorithm is due to its improper implementation.

3.1 Artificial Bee Colony

ABC is a swarm intelligence based CI technique which imitates the intelligent behavior of honey bee swarm in determining the food sources. A potential solution of the optimization problem is referred to as food source and the fitness of the solution is referred as the nectar amount in the food source. Three types of bees are used in ABC to search the optimal solution (i) employed bees, (ii) the onlooker bees and (iii) the scout bees. In the first stage, the employed bees determine new food sources based on the food sources that it had visited previously and on the information received from other employed bees. The employed bee changes its position if it is able to determine a better food source. In the second stage, the onlooker bees determine new food sources similar to the employed bees but also use the information about the nectar content of food sources from the employed bees. The onlooker bees update their positions if they are able to determine a better food source than their previous food source. The solution, which repeatedly fails a specified number of times to determine a better solution in the employed bee phase or the onlooker phase, is replaced by a random solution in the scout phase of the algorithm. These three stages are repeatedly performed until the termination criterion is achieved. The scout bees are reported to explore the search space, as it does not utilize the information about the previous position whereas the onlooker and employed bees are reported to perform the exploitation of the search space. The food source discovered with the largest nectar content is considered as the best solution. The critical issues in the implementation of ABC are discussed in (Mernik, Liu et al. (2015)). In this work, we have used the implementation of ABC that is publicly available (https://goo.gl/zqeujq).

3.2 Dynamic Neighborhood Learning based Particle Swarm Optimizer (DNLPSO)

PSO is one among the most popular swarm intelligence technique and a large number of variants have been proposed to improve its performance. In PSO, the solutions are termed as particles and new potential solutions are determined using the historical best of a particular particle, (known as personal best), as well as the global best of all the particles. Comprehensive Learning PSO was proposed (Liang, Qin et al. (2006)) to increase the explorative nature of the algorithm by (i) removing the dependency on the global best and (ii) relying on the personal best of other solutions rather than its own personal best. However, it has been reported that this leads to a reduction in the convergence rate. Dynamic Neighborhood Learning based Particle Swarm Optimizer (DNLPSO) (Nasir, Das et al. (2012)) is a modified version of CLPSO in which each particle updates its velocity with the help of its personal best or the exemplar particle that is selected from its neighborhood and also the global best. The formation of neighborhood with a ring topology has been recommended though other topologies can also be implemented. The learning from the neighborhood solution is implemented only if the personal best does not improve for a specified number (refreshing gap) of consecutive iterations. In order to encourage exploration, a dynamic neighborhood is employed which is formed after a specified number of iterations. In this work, we have employed the implementation of DNPLSO provided by the authors (https://goo.gl/A8npGD).

3.3 Genetic Algorithm (GA)

GA (Deb (2001)) is probably the most popular and one of the earliest evolutionary techniques. It is based on the working principles of natural genetics and natural selection. The search for an optimal solution begins with randomly generated solutions within the search space that constitutes the population. Each solution in the population is termed as chromosome whose fitness is represented by the value of the objective function. In every generation, reproduction operators are used to form the mating pool by choosing multiple copies of better chromosomes. Variation operator such as crossover or mutation are employed to generate offspring which are potential solutions. The occurrence of crossover or mutation of a chromosome is decided based on the values of user defined parameters such as crossover and mutation probability. The reproduction, crossover and mutation operators perform the exploration and exploitation of the search space. At the end of every generation, best chromosomes among the parents and offspring progress to the next generation. In this work, we have used the inbuilt *ga* function of MATLAB2016a.

3.4. Multi-Population Ensemble Differential evolution (MPEDE)

DE is also a popular evolutionary technique that has found applications in diverse areas. It can employ multiple mutation strategies and it is not always possible to select the best mutation strategy for an arbitrary problem. A variant of DE, MPEDE (Wu, Mallipeddi et al. (2016)) embeds three mutation strategies in its working. In particular, the population is divided into four sub groups in every generation, three of which are equal in size and are known as the indicator population whereas the remaining members constitute the reward population that is comparatively larger in size. Each indicator population is assigned a different mutation strategy (Current-to-pbest/1" and "rand/1" with the uniform binary crossover and Current-to-rand/1 without any crossover). It also employs an adaptive strategy to tune various parameters required in the mutation strategies. After periodic intervals, the best performing strategy, determined with respect to the improvement in fitness value and the number of functional evaluations utilized, is assigned to the reward population. The implementation of MPEDE provided by the authors is used in this work (https://goo.gl/A8npGD).

3.5. Sanitized Teaching-Learning Based Optimization (s-TLBO)

TLBO is a population based stochastic technique and has been proposed multiple times in literature. It is inspired from the concept of knowledge transfer in a classroom environment through the teaching and learning process. A solution is termed as student and the marks scored in various subjects offered to the student is analogous to the values of the design variables in the optimization problem. A set of solutions that constitutes the population is known as the class. The solution that has the best fitness is termed as teacher. Each iteration of TLBO involves two phases, viz., teacher phase and learner phase. In the teacher phase, a potential solution is generated for every member of the class based on the teacher, marks of the particular member, teaching factor and mean of the class. In the student phase, a potential solution is generated for every member of the class depending on the marks of the particular member and marks of a randomly selected learning partner. In the teacher phase as well as the students phase, a new solution is accepted if it is better than the solution undergoing the teaching-learning process. In view of the various discrepancies reported in literature (Črepinšek, Liu et al. (2012)), we have developed an in-house code which does not remove any duplicates and provides a deterministic relation between the maximum number of functional evaluations and the number of iterations in s-TLBO (https://goo.gl/RGsbNM).

4 Experimental Settings

We have considered ten instances of job shop scheduling problems that have been widely used in literature (Jain and Grossmann (2001), Kotecha, Bhushan et al. (2011), Maharana and Kotecha (2019)) These instances arise from five problems with the smallest problem containing 3 orders and 2 machines and the largest problem comprises of 20 orders and 5 machines. The details of the five problems are provided in Table 1. Each of the five problem have two datasets arising from the difference in processing time. The ten instances are labelled as P1S1, P1S2 and so on till P5S2 in which 'P1' corresponds to Problem 1 and 'P5' corresponds to Problem 5 whereas 'S1' and 'S2' indicate to the two datasets of a particular problem. For the sake of brevity, the due date, release date, processing time and cost are provided in Table 2 for Problem 1. For the rest of the eight instances, the data can be obtained from the literature (https://goo.gl/QPM0G4). The first instance indicated by 'S1' have longer processing times and have been reported to be more complex as these contain fewer feasible schedules than the second instance S2.

Very often there are conflicting claims about the performance of CI techniques which are primarily attributed

to improper implementation of the algorithm and the absence of the details regarding the values of algorithm specific parameters. The details of the various algorithm specific parameters are provided in Table 3 and are primarily based on the recommendations in the literature by their authors. In view of the stochastic nature of the metaheuristic techniques, 15 independent runs are executed for each of the ten instances. Hence a total of 750 unique trials (5 Problems x 2 datasets x 15 runs x 5 algorithms) are executed to evaluate the performance of the five algorithms. All the simulations were performed on MATLAB 2016a on an Intel i7 processor (3.6 GHz) PC with 16 GB RAM.

The population size N_p for each algorithm is a function of the number of decision variables of the problem and is set as $20D$, where D is the number of decision variables of the problem (given in Table 1). For a fair comparison between the algorithms, the termination criterion for all the algorithms is set as the maximum allowed functional evaluations (*MaxFE*). The inbuilt *ga* function of MATLAB requires $N_p(N_g+1)+1$ functional evaluation for performing N_g generations where N_p is the population size. The *ga* function is used for 500 generation and hence the maximum number of functional evaluation for all the algorithms is given by

$$MaxFE = 501N_p + 1 \tag{10}$$

CI techniques are primarily designed to solve optimization problem with continuous decision variables. However, the decision variable corresponding to the machine used by a particular order has to be an integer. In order to accommodate these integer variables, the rounding scheme is employed as shown in Eq. (11)

$$\begin{aligned} y_i &= \lfloor y_c \rfloor \quad if\ y_c - \lfloor y_c \rfloor < 0.5, \\ y_i &= \lceil y_c \rceil \quad otherwise \end{aligned} \tag{11}$$

where y_c is the continuous variable determined by the algorithm and y_i is the integer value determined using this continuous variable. However in the case of GA, the inbuilt function of MATLAB permits the use of integer variables and the same has been used to handle integer variables.

5. Results and Discussions

In this section, we evaluate the performance of five different stochastic population based CI techniques for solving the combinatorial job shop scheduling problems. The performance of the algorithms in the individual runs is shown in Fig. 1. In P1S1, it can be seen that all the algorithms except DNLPSO are able to determine the best solution. In P1S2, it can be seen that the algorithms are able to either obtain the objective function value of 18 or 21. In P5S1, most of the runs of the algorithms are unable to reach the best solution and are clustered around the value of 400, while a few runs of MPEDE being able to reach the best solution. It can also be observed that DNLPSO is consistently unable to determine the best solution for any of the ten instances. Fig. 2 shows the number of runs in which an algorithm was unable to determine a feasible solution. It can be observed that the number of infeasible runs in the dataset 1 is higher than for dataset 2. This can be attributed to the reason that the processing time in second dataset is smaller and thus the number of feasible solutions are higher. In P1S2 and P2S2, all the algorithms are able to determine a feasible solution in all the runs. MPEDE has the lowest number of infeasible runs among all the algorithms followed by ABC.

The best, mean and standard deviation values of the best objective function value obtained in the fifteen runs by each algorithm are shown in Table 4 and Table 5. Table 4 also shows the results of Grey-Wolf Optimization and the JAYA algorithm which have been reported in recent literature (Maharana and Kotecha (2019)) with identical termination conditions as used in this work. In Table 4, the values shown for GA (MATLAB) have been generated as described earlier whereas the value indicated against GA correspond to those provided in the literature (Kotecha, Bhushan et al. (2011)) which does not clearly specify the termination criteria. It can be observed that the best value as well as the mean value obtained by all the algorithms for P1S1 are identical except for DNLPSO. Moreover, the value of zero to standard deviation implies that the algorithms are able to determine the global optima of this problem in all the runs. In the case of P1S2, all the algorithms except DNLPSO are able to determine the best value. Moreover, ABC is able to determine the solution in all the runs whereas s-TLBO showed the largest standard deviation. Thus in the case of P1, ABC seems to outperform all the other algorithms as it is able to consistently determine the best solution. The convergence curve of each algorithm for best run of P1 is shown in Fig. 3. The values in y-axis shows the best solution discovered until the completion of the specified number of functional evaluations denoted by the value in x-axis. It can be observed that all the algorithms are able to determine an identical solution in both instances except DNLPSO in P1S2. For all the

problems, the Gantt chart is shown for the best solution that has been determined across the five algorithms. Fig. 4 shows the Gantt chart for the two instances of P1. It can be observed that the processing time of the orders in the second instance are longer in P1S1 as compared to P1S2.

Similar to P1, ABC outperforms the other algorithms with its ability to determine the better solution in both the instances of P2. Moreover, as reflected in the mean values, it is able to consistently determine better solutions. MPEDE is also able to determine the best solution reported by ABC and has a better mean than all algorithms except ABC. Across all algorithms, the standard deviation is high in P2S1. The performance of GA and s-TLBO are almost identical and these have not been able to determine the best solution of 44 even in one of their runs. The performance of DNLPSO is not satisfactory. The time required for completing all the orders in both the instances is 28 time units and corresponds to the due date of order 4. The convergence curve corresponding to the best run of every algorithm for P2 is shown in Fig. 5. It can be observed that in P2S1, ABC requires larger number of functional evaluations to determine the best solution. In both the instances, MPEDE shows the fastest convergence to the best solution. The Gantt chart for P2 is shown in Fig. 6.

In P3S1, the best value reported by GWO is better than all the five algorithms considered in this work. It is observed that s-TLBO reports the best solution of 103 for P3S1. Though the best solution reported by MPEDE is 104, it has a very impressive mean of 170.93 against the mean of 216.80 determined by s-TLBO and 217.90 determined by GWO. The performance of the rest of the three algorithms is not satisfactory as their best value is more than twice that of s-TLBO and MPEDE. In the case of P3S2, MPEDE and s-TLBO are able to determine an identical best solution with MPEDE being able to determine a better mean and a lower standard deviation. Fig. 7. shows the convergence curve of the algorithms for P3. In both the instances, s-TLBO is able to converge faster than MPEDE to a better solution. The Gantt chart for P3 is given in Fig. 8. For P3S1, it can be observed that orders 4, 5, 6, 8 and 10 are processed on M1, the orders 2, 3, 9 and 12 are processed on M2 whereas the remaining three orders are processed on M3. In the case of P3S2, M1 is not used to process any order whereas eight orders are processed on M2 and four orders are processed on M3.

In P4S1, it can be observed that GWO was able to determine a better solution (best) compared to the identical solution of 119 determined by s-TLBO and MPEDE. However, the mean and standard deviation of MPEDE is significantly better than that of GWO and TLBO. Among the other algorithms, ABC performs better than GA and DNLPSO. In P4S2, MPEDE is able to determine the best solution and has a better mean than s-TLBO. However, the difference in performance for this problem is not as prominent as P4S1. In Fig. 9., it can be observed that all the algorithms have converged, with MPEDE and s-TLBO providing an identical solution for P4S1 whereas MPEDE converges to a better solution than s-TLBO in P4S2. The Gantt chart for P4 is shown in Fig. 10. For the first instance, all the machines are used whereas only three machines are used in the second instance. In both instances, all the orders are completed at 38 with order 9 being the last processed order.

In P5S1, it can be observed that the performance of MPEDE is significantly better than all the algorithms including s-TLBO which was otherwise performing reasonably similar to MPEDE in the other instances. It should be noted that the best solution determined by MPEDE and GWO are identical. Though the standard deviation of MPEDE is higher than s-TLBO and GWO, the mean of MPEDE is better than that of both algorithms. In P5S2, the best solution determined by s-TLBO is better than that determined by MPEDE but the mean determined by MPEDE is better than that of s-TLBO. As in all the other problems, the performance of DNLPSO is the least satisfactory. Fig.11. shows the convergence curve for P5 and it can be observed that some of the algorithms have not converged but have been terminated due to the utilization of the maximum number of functional evaluations. Unlike in other problems, MPEDE utilizes more than 80% of the functional evaluations to determine the best solution. Fig.12. shows the Gantt chart for P5 and it can be observed that all the 20 orders are scheduled without any conflict on any of the machines. Moreover, all the orders are processed accordance to their release date and are completed by the due date. On analyzing the overall performance of algorithms on this scheduling problem, it is observed that MPEDE has consistently determined better solutions in majority of runs except in two smaller problems (P1S2 and P2S1).

5.3 Time Complexity:

Time complexity provides a measure of the time required by an algorithm to solve the optimization problem. In this work, we determine the time complexity of the algorithm on every problem instance as per the following procedure that is predominantly based on the procedure used to benchmark computational intelligence techniques in the competitions held at the Congress on Evolutionary Computation.

Step 1: Determine the time required (T_M) for computing certain basic mathematical operations for 10^6 times.

Step 2: Determine the time required (T_O) to evaluate the objective function for MaxFe times.
Step 3: Determine the time required (T_A) for an algorithm to solve a particular instance (with MaxFe as the termination criterion). In view of the stochastic nature of the algorithms, T_A is determined for five runs and its mean (T_{Av}) is used to evaluate time complexity.
Step 4: The time complexity of an algorithm is given by $(T_{Av} - T_O)/ T_M$.

It should be noted that Step 3 is to be performed for every combination of the algorithm and problem instance whereas Step 2 is independent of the algorithm and is to be executed for every problem instance. The time complexity for the 50 combinations (5 algorithms x 2 datasets x 5 problems) is depicted in Fig. 13. Across all algorithms, it can be observed that the time complexity is higher for the second dataset than the first dataset. Among the algorithms, DNLPSO possesses the lowest time complexity whereas the inbuilt *ga* function of MATLAB possesses the highest time complexity. However it should be noted that DNLPSO was not able to determine the best solution. The best solution is usually determined by either MPEDE or s-TLBO and it can be observed that MPEDE has a lower time complexity than s-TLBO. Moreover, the increase in the time complexity with an increase in the problem dimension is relatively lower for MPEDE.

5. Conclusion

The performance of five recent optimization techniques has been evaluated on the dissimilar job shop scheduling problems and it was observed that MPEDE was able to consistently determine the best solution followed by s-TLBO even for the larger problems. It was also observed that the first dataset (S1) which involves higher processing time are comparatively challenging to solve due to the presence of only few feasible solutions. DNPLSO showed relatively poor performance but was able to generate better solutions in its initial stage. The time complexity of MPEDE does not significantly scale up with an increase in the size of the problem. Future work can involve design of efficient schemes as well as hybridization among various algorithms to efficiently determine the optimal solution.

References

Allahverdi, A., Ng C. T., Cheng T. C. E. and Kovalyov M. Y. (2008), A survey of scheduling problems with setup times or costs, European Journal of Operational Research 187(3), 985-1032.

Chauhan, S. S. and Kotecha P. (2018), An efficient multi-unit production planning strategy based on continuous variables, Applied Soft Computing, 68, 458-477.

Chauhan, S. S., Sivadurgaprasad C., Kadambur R. and Kotecha P. (2018), A novel strategy for the combinatorial production planning problem using integer variables and performance evaluation of recent optimization algorithms, Swarm and Evolutionary Computation, DOI https://doi.org/10.1016/j.swevo.2018.04.004

Chung, S. H., Tai Y. T. and Pearn W. L. (2009), Minimising makespan on parallel batch processing machines with non-identical ready time and arbitrary job sizes, International Journal of Production Research, 47(18), 5109-5128.

Črepinšek, M., Liu S. H. and Mernik L. (2012), A note on teaching learning-based optimization algorithm, Information Sciences, 212, 79-93.

Deb, K. (2001), Multi-objective optimization using evolutionary algorithms, *ISBN 978-81-265-2804-2*, John Wiley & Sons, Inc. New York, NY, USA

Dobson, G. and Arai Yano C. (1994), Cyclic scheduling to minimize inventory in a batch flow line, European Journal of Operational Research, 75(2), 441-461.

Eskandar, H., Sadollah A., Bahreininejad A. and Hamdi M. (2012), Water cycle algorithm – A novel metaheuristic optimization method for solving constrained engineering optimization problems, Computers & Structures, 110-111, 151-166.

Hasda, R. K., Bhattacharjya R. K. and Bennis F. (2017), Modified genetic algorithms for solving facility layout problems, International Journal on Interactive Design and Manufacturing, 11(3), 713-725.

Hooker, J. N. (2005), A Hybrid Method for the Planning and Scheduling, Constraints, 10(4), 385-401.

Hooker, J. N., Ottosson G., Thorsteinsson E. S. and Kim H. J. (1999), On Integrating Constraint Propagation and Linear Programming for Combinatorial Optimization, Proceedings of 16th National Conference on Artificial Intelligence. Orlando, Florida, MIT Press, 136-141.

Jain, V. and Grossmann I. E. (2001), Algorithms for Hybrid MILP/CP Models for a Class of Optimization Problems, Informs Journal on Computing, 13(4), 258-276.

Kashan, A. H. and Karimi B. (2009), A discrete particle swarm optimization algorithm for scheduling parallel machines, Computers & Industrial Engineering, 56(1), 216-223.

Kashan, A. H., Karimi B. and Jenabi M. (2008), A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes, Computers & Operations Research, 35(4), 1084-1098.

Kotecha, P. R., Bhushan M. and Gudi R. D. (2011), Constraint programming and genetic algorithm, Stochastic Global Optimization, World scientific, 2, 619-675, DOI https://doi.org/10.1142/9789814299213_0018

Lee, W.C., Wu C.C. and Chen P. (2006), A simulated annealing approach to makespan minimization on identical parallel machines, The International Journal of Advanced Manufacturing Technology, 31(3), 328-334.
Lenstra, J. K., Shmoys D. B. and Tardos É. (1990), Approximation algorithms for scheduling unrelated parallel machines, Mathematical Programming, 46(1), 259-271.
Liang, J. J., Qin A. K., Suganthan P. N. and Baskar S. (2006), Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Transactions on Evolutionary Computation, 10(3), 281-295.
Liu, B., Wang L. and Jin Y. H. (2007), An Effective PSO-Based Memetic Algorithm for Flow Shop Scheduling, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 37(1), 18-27.
Maharana, D. and Kotecha P. (2019), Optimization of job shop scheduling problem with Grey Wolf Optimizer and JAYA algorithm, Smart Innovations in Communication and Computational Sciences, Springer, DOI 10.1007/978-981-10-8968-8
Mastrolilli, M. and Gambardella L. M. (2000), Effective neighbourhood functions for the flexible job shop problem, Journal of Scheduling, 3(1), 3-20.
Mernik, M., Liu S. H., Karaboga D. and Črepinšek M. (2015), On clarifying misconceptions when comparing variants of the Artificial Bee Colony Algorithm by offering a new implementation, Information Sciences, 291, 115-127.
Mirjalili, S. (2015), Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm, Knowledge-Based Systems, 8, 228-249.
Moslehi, G. and Mahnam M. (2011), A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search, International Journal of Production Economics, 129(1), 14-22.
Nasir, M., Das S., Maity D., Sengupta S., Halder U. and Suganthan P. N. (2012), A dynamic neighborhood learning based particle swarm optimizer for global numerical optimization, Information Sciences, 209, 16-36.
Pezzella, F., Morganti G. and Ciaschetti G. (2008), A genetic algorithm for the Flexible Job-shop Scheduling Problem, Computers & Operations Research, 35(10), 3202-3212.
Pfund, M., Fowler J. W. and Gupta J. N. D. (2004), A survey of algorithms for single and multi-objective unrelated parallel-machine deterministic scheduling problems, Journal of the Chinese Institute of Industrial Engineers, 21(3), 230-241.
Punnathanam, V. and Kotecha P. (2016), Yin-Yang-pair Optimization: A novel lightweight optimization algorithm, Engineering Applications of Artificial Intelligence, 54, 62-79.
Wu, G., Mallipeddi R., Suganthan P. N., Wang R. and Chen H. (2016), Differential evolution with multi-population based ensemble of mutation strategies, Information Sciences, 329, 329-345.
Xia, W. and Wu Z. (2005), An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems, Computers & Industrial Engineering, 48(2), 409-425.

List of Figures

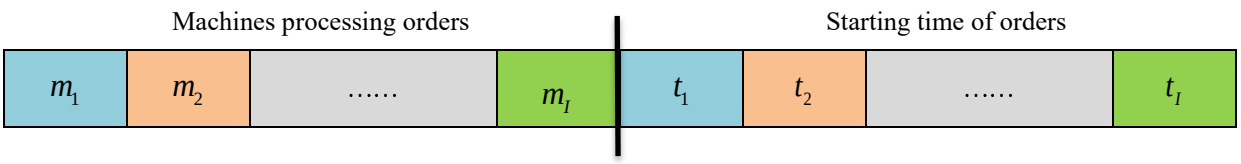

Fig. 1. Example of a solution vector

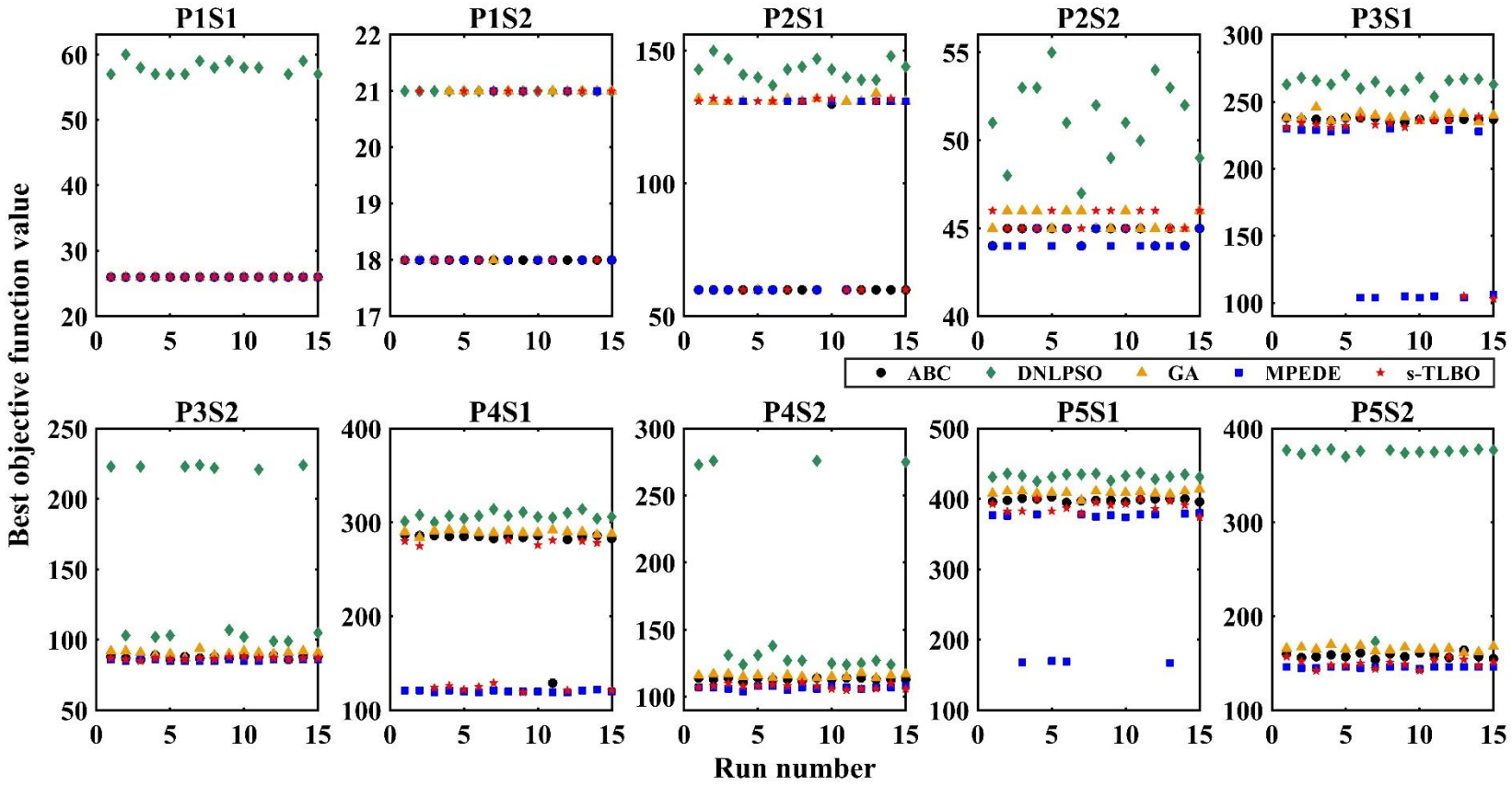

Fig. 2. Performance of the algorithms in the various runs

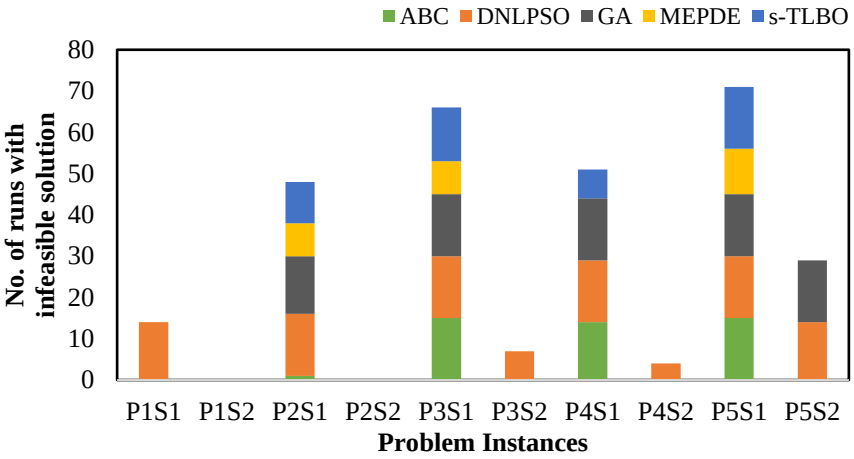

Fig. 3. Details of infeasible runs

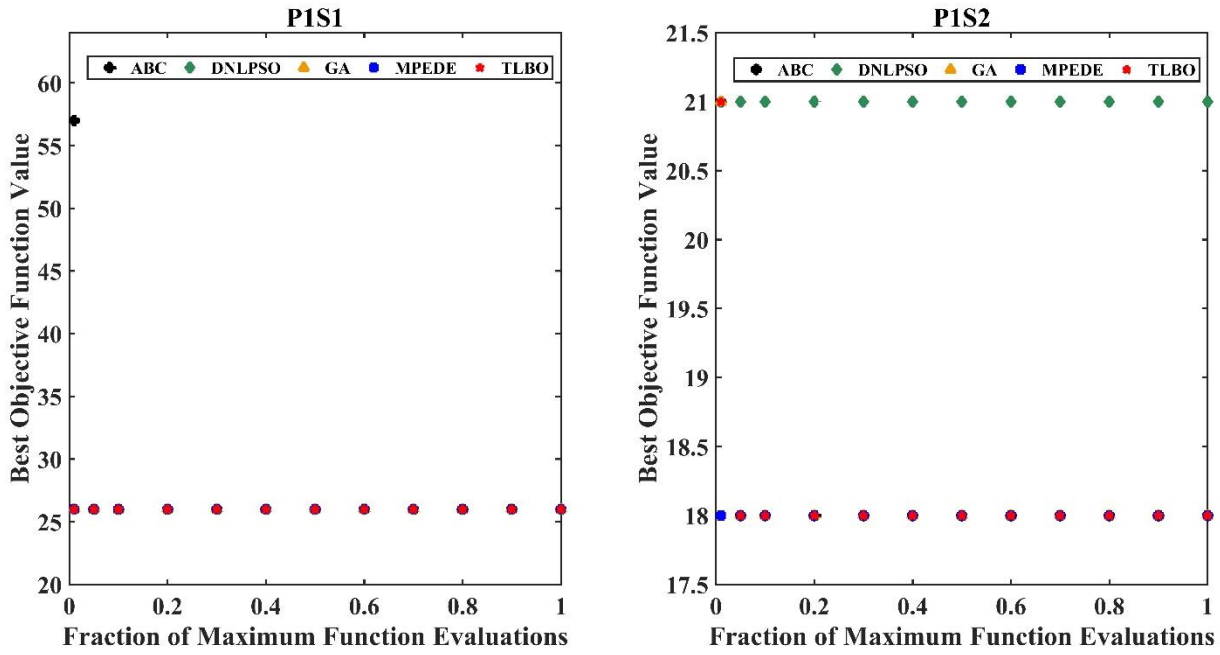

Fig. 4. Convergence curves for Problem 1

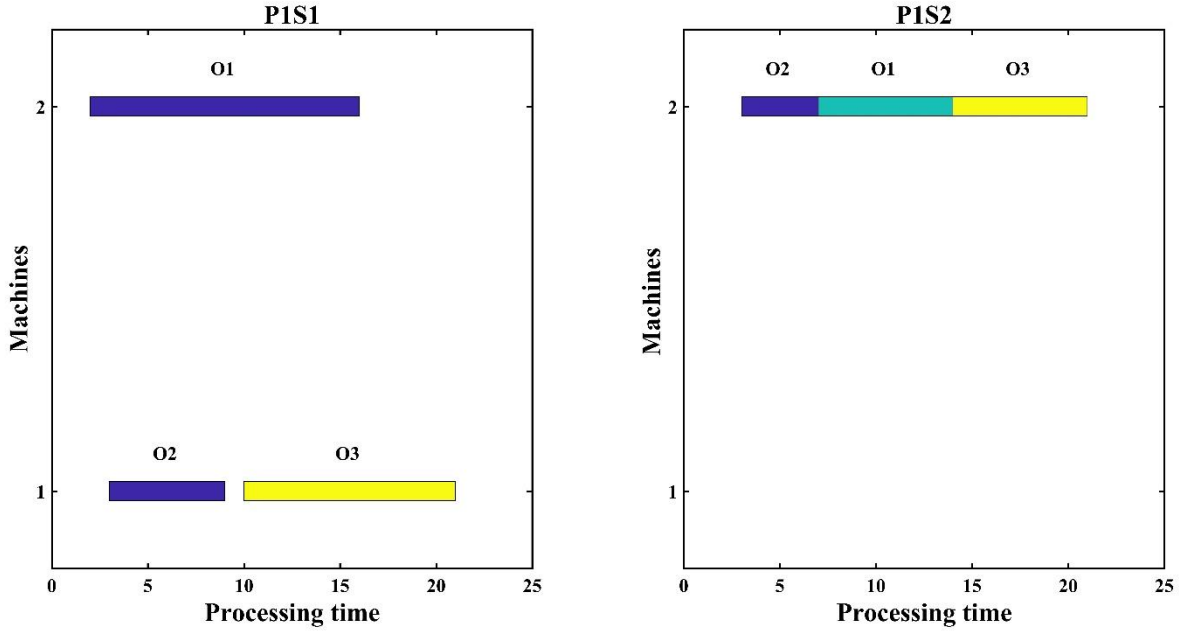

Fig. 5. Gantt chart for Problem 1 (3 orders and 2 machines)

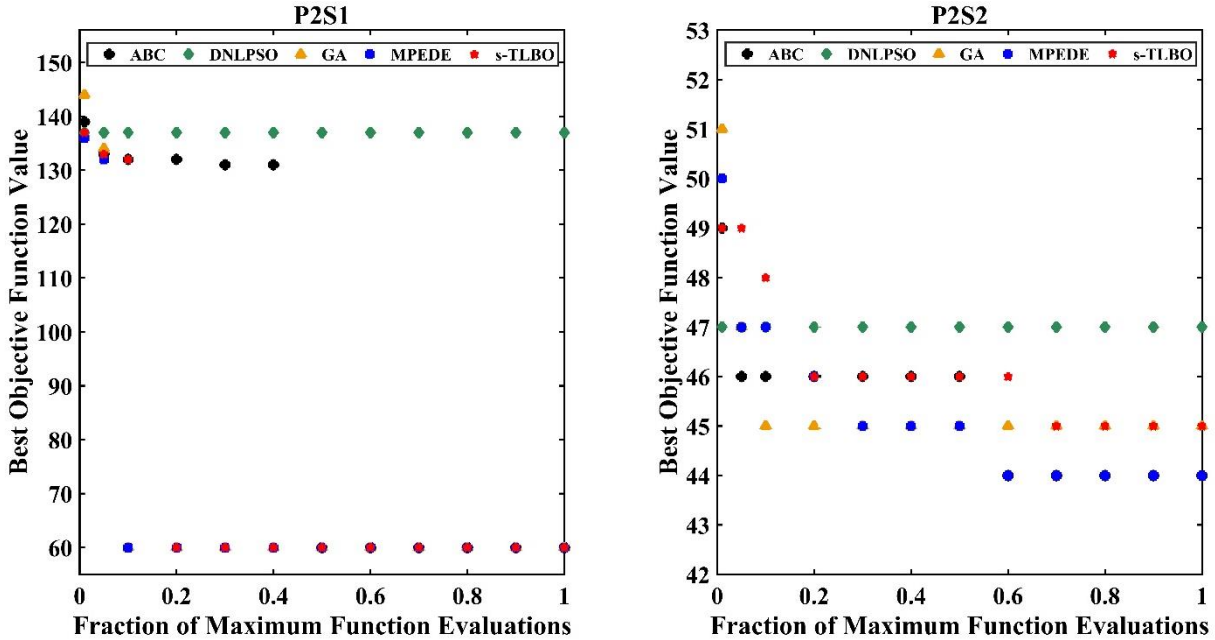

Fig. 6. Convergence curves for Problem 2

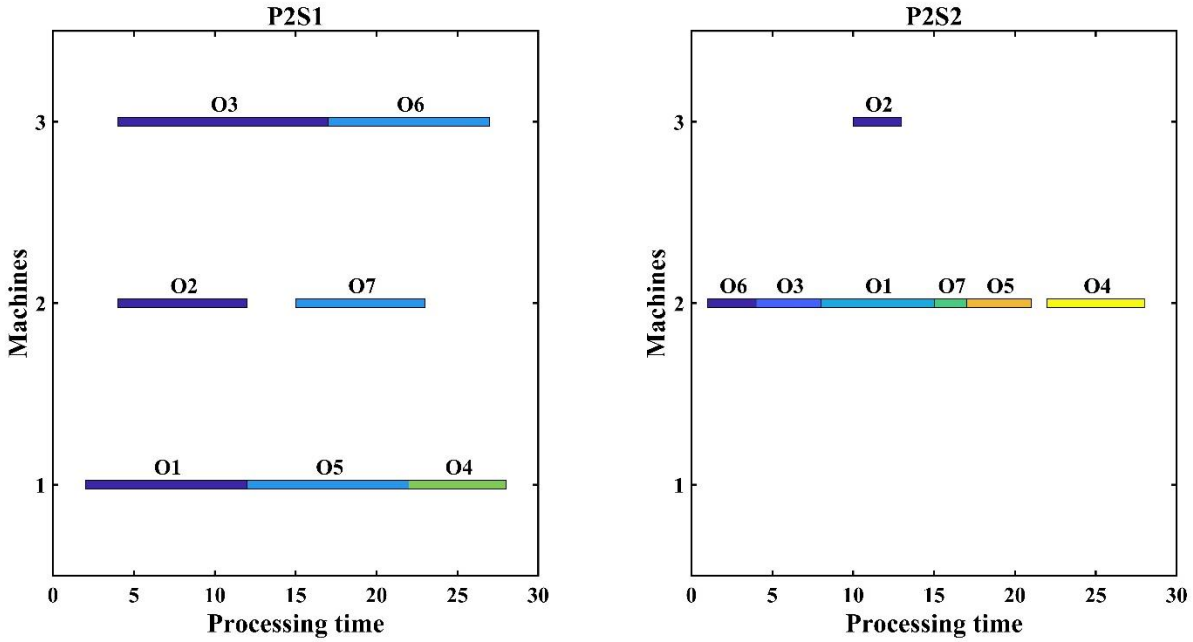

Fig. 7. Gantt chart for Problem 2 (7 orders and 3 machines)

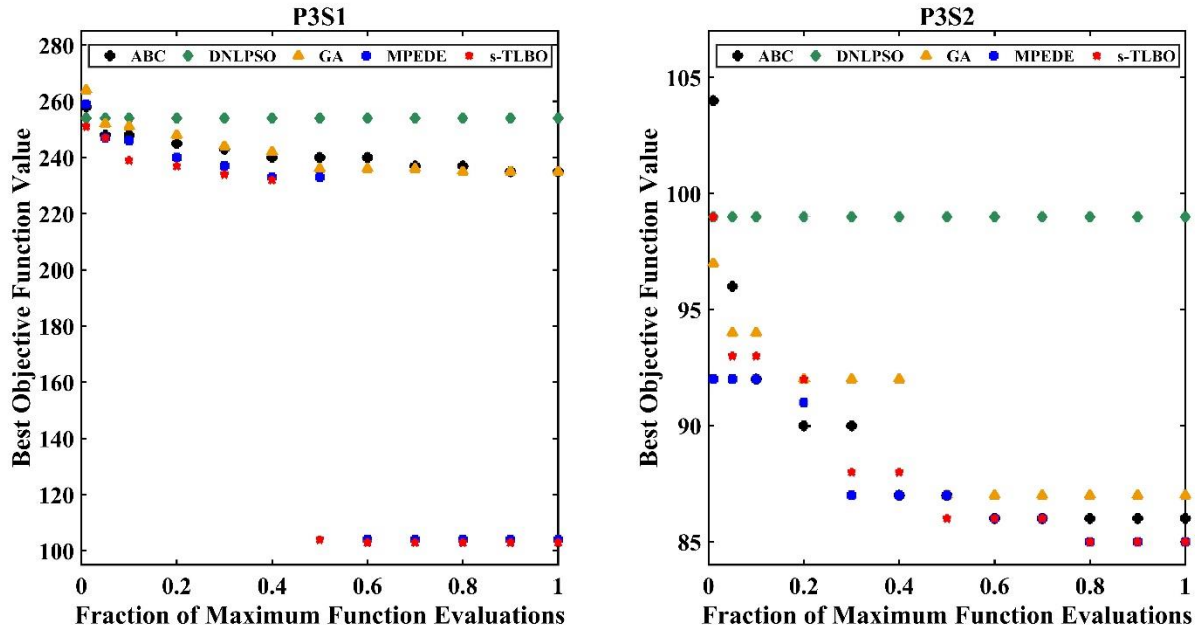

Fig. 8. Convergence curves for Problem 3

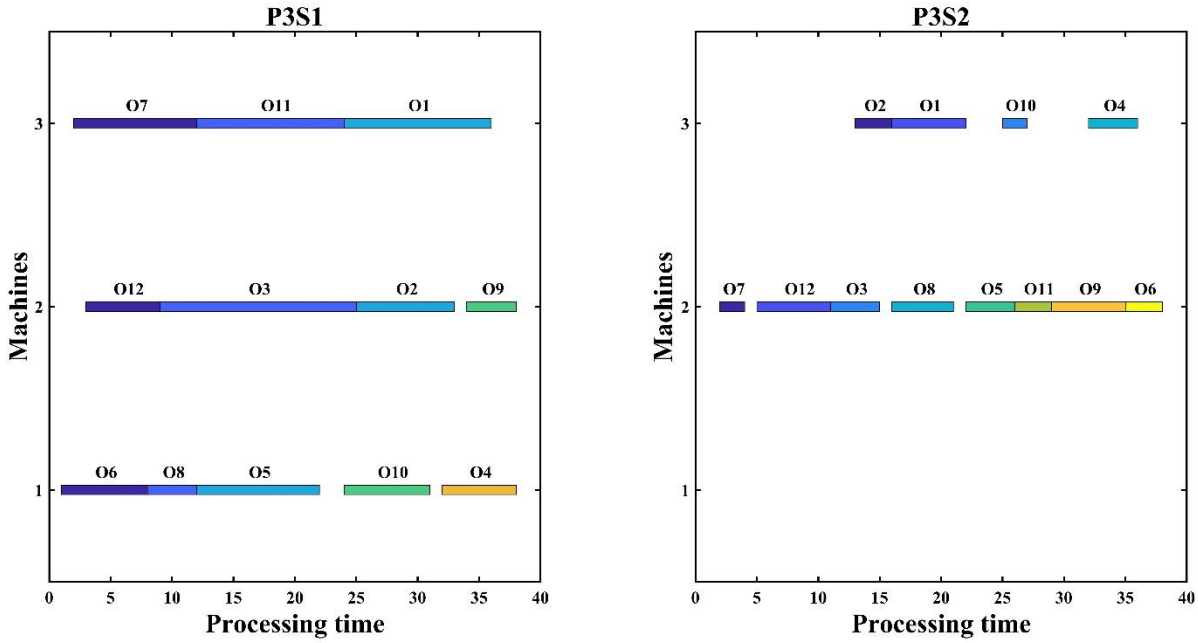

Fig. 9. Gantt chart for Problem 3 (11 orders and 3 machines)

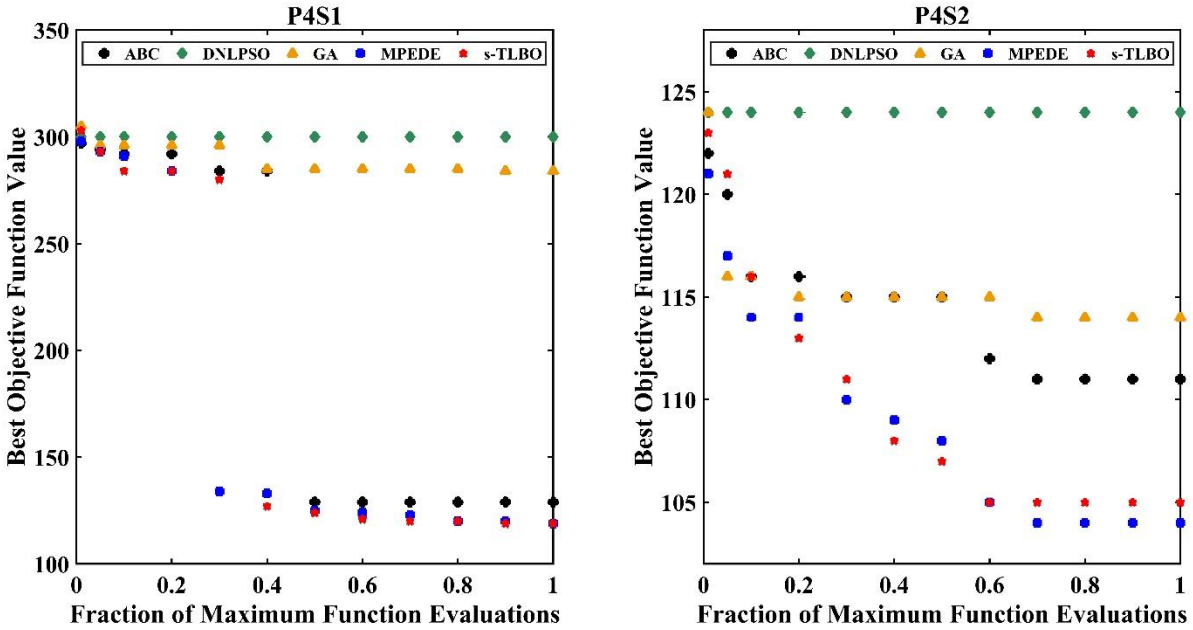

Fig. 10. Convergence curves for Problem 4

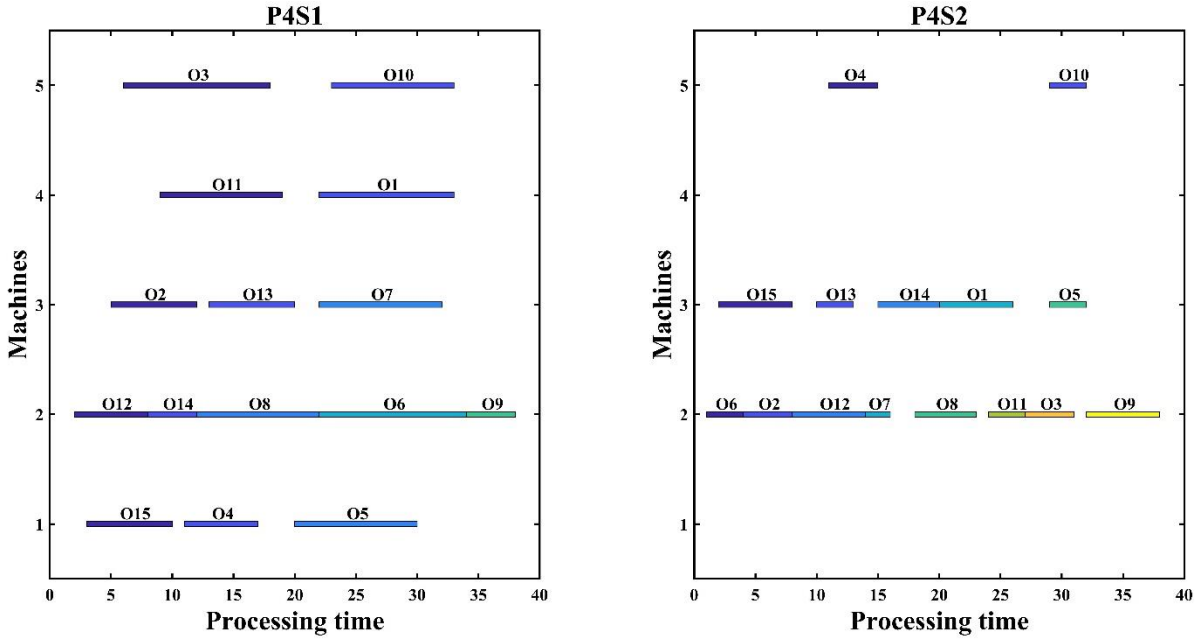

Fig. 11. Gantt chart for Problem 4 (15 orders and 5 machines)

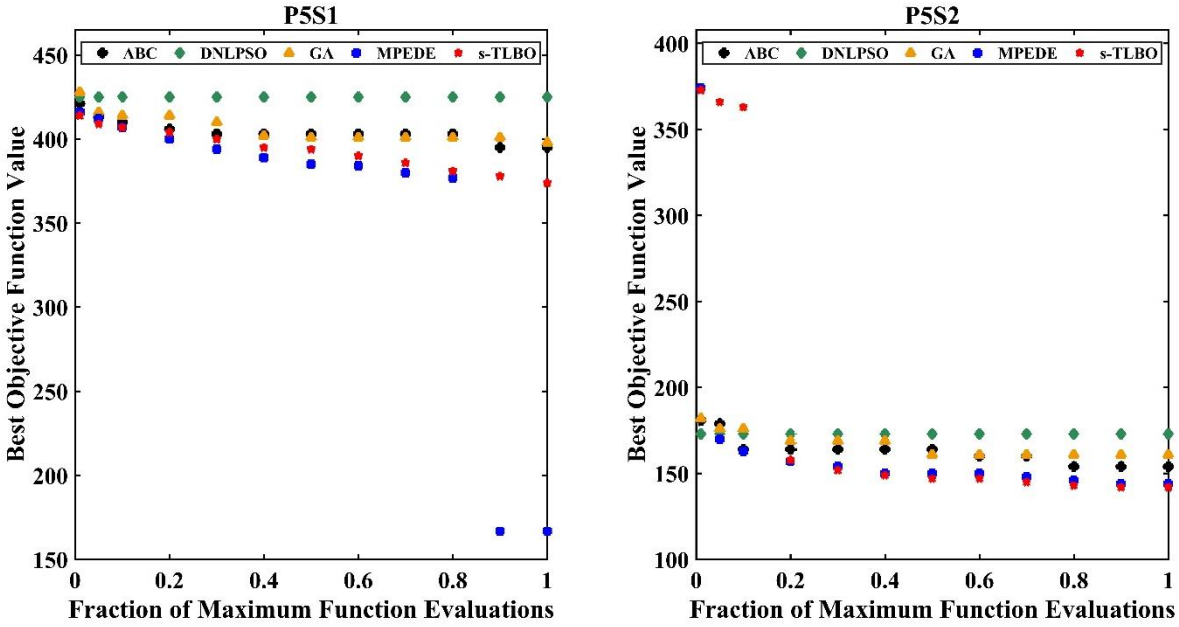

Fig. 12. Convergence curves for Problem 5

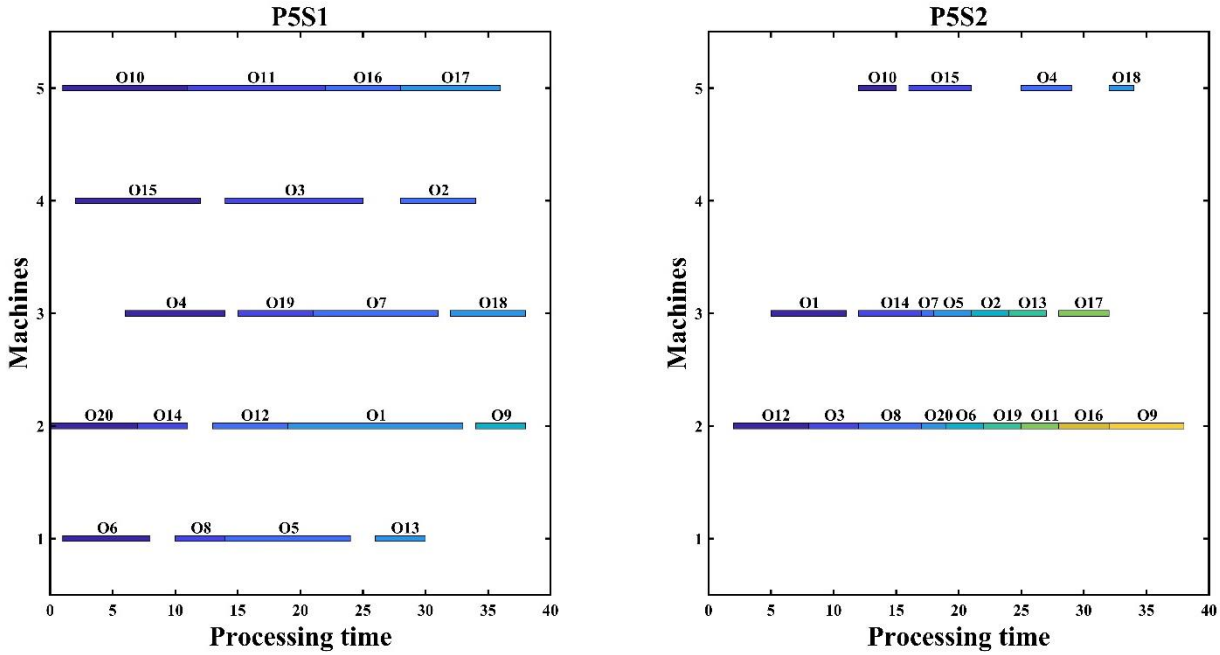

Fig. 13. Gantt chart for Problem 5 (20 orders and 5 machines)

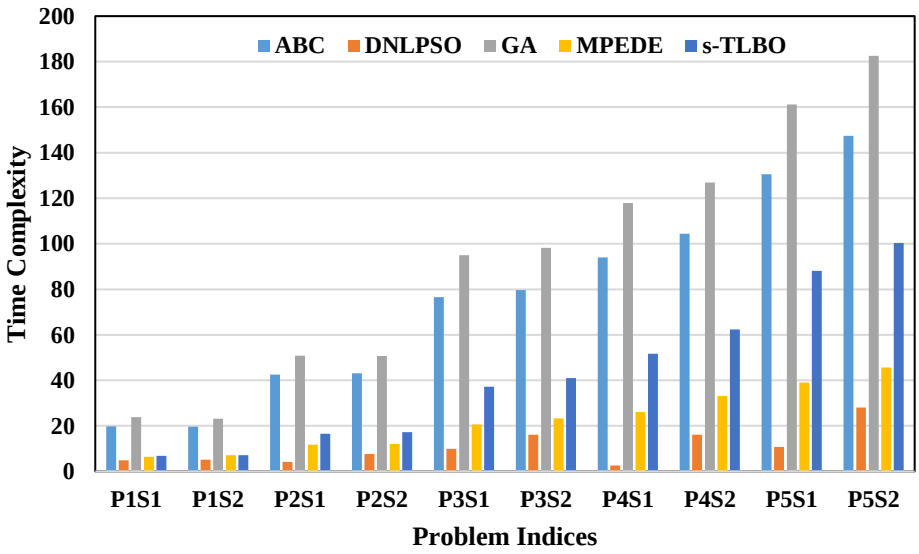

Fig.14. Time complexity of the algorithms

List of Tables

Table 1. Details of orders and machines for all the problems
Table 2. Data for Problem 1
Table 3. Parameter settings for various algorithms
Table 4. Statistical analysis of best objective function value obtained in Problem 1 to Problem 3
Table 5. Statistical analysis of best objective function value obtained in the Problem 4 to Problem 5

Table 1. Details of orders and machines for all the Problems

Problem	Number of orders	Number of Machines	Number of decision variables
P1	3	2	6
P2	7	3	14
P3	12	3	24
P4	15	5	30
P5	20	5	40

Table 2. Data for Problem 1 (Jain and Grossmann (2001))

Order(i)	r_i	d_i	Cost on machine		Durations			
					Dataset 1		Dataset 2	
			M1	M2	M1	M2	M1	M2
1	2	16	10	6	10	14	5	7
2	3	13	8	5	6	8	3	4
3	4	21	12	7	11	16	5	7

Table 3. Parameter settings for various algorithms

Algorithms	Parameters	Values
ABC	Number of onlooker bees	Population size
	Abandonment Limit Parameter	*round(0.6 X Population size X Problem Dimension)*
	Upper bound of acceleration coefficient	1
DNLPSO	Acceleration coefficients	1.49445
	w_0	0.9
	w_1	0.4
	Refreshing group	3
	Regrouping gap	5
	Learning probability	$0.45-0.4\frac{(Population\ Index-1)}{(Population\ size-1)}$
	Parameter for determination of velocity bound	2
MPEDE	Fraction of indicator population for each strategy	0.2
	Best mutation strategy determining period	20

Table 4. Statistical analysis of best objective function value obtained in Problem 1 to Problem 3

Problems	Algorithms	Best	Worst	Mean	Median	SD
P1S1	ABC	26	26	26	26	0
	DNLPSO	26	60	55.8	58	8.3
	GA (MATLAB)	26	26	26	26	0
	MPEDE	26	26	26	26	0
	TLBO	26	26	26	26	0
	GWO	26	–	26	26	0
	JAYA	26	–	26	26	0
	GA	26	26	26	–	0
P1S2	ABC	18	18	18	18	0
	DNLPSO	21	21	21	21	0
	GA (MATLAB)	18	21	20	21	1.46
	MPEDE	18	21	18.8	18	1.37
	TLBO	18	21	19.8	21	1.52
	GWO	18	–	18	18	0
	JAYA	18	–	18	18	0
	GA	18	21	18.72	–	1.28
P2S1	ABC	60	130	64.67	60	18.07
	DNLPSO	137	150	143	143	3.76
	GA (MATLAB)	60	134	126.67	131	18.46
	MPEDE	60	131	97.87	131	36.66
	TLBO	60	132	107.6	131	34.84
	GWO	60	–	117.9	132	28.9
	JAYA	60	–	117.9	132	28.9
	GA	–	–	–	–	–
P2S2	ABC	44	45	44.73	45	0.46
	DNLPSO	47	55	51.2	51	2.27
	GA (MATLAB)	45	46	45.47	45	0.52
	MPEDE	44	45	44.33	44	0.49
	TLBO	45	46	45.47	45	0.52
	GWO	44	–	45.7	46	1.3
	JAYA	44	–	45.7	46	0.6
	GA	–	–	–	–	–
P3S1	ABC	235	238	236.93	237	0.96
	DNLPSO	254	270	263.8	265	4.43
	GA (MATLAB)	235	246	239.13	239	2.75
	MPEDE	104	230	170.93	228	64.26
	TLBO	103	239	216.8	233	45.87
	GWO	102	–	217.9	233	42.3
	JAYA	233	–	237.4	237	2.5
	GA	–	–	–	–	–
P3S2	ABC	86	89	87.53	88	0.99
	DNLPSO	99	224	158.67	107	62.19
	GA (MATLAB)	87	94	90.73	91	1.67
	MPEDE	85	86	85.53	86	0.52
	TLBO	85	88	86.67	87	0.82
	GWO	86	–	90.1	90	1.8
	JAYA	88	–	90.2	90	1.3
	GA	–	–	–	–	–

Table 5. Statistical analysis of best objective function value obtained in the Problem 4 to Problem 5

Problems	Algorithms	Best	Worst	Mean	Median	SD
P4S1	ABC	129	287	274.47	285	40.26
	DNLPSO	300	314	306.93	307	4.08
	GA (MATLAB)	284	292	289.53	290	2.13
	MPEDE	119	122	120.2	120	0.94
	TLBO	119	281	195.87	129	80.27
	GWO	118	–	147	123	57.1
	JAYA	134	–	282.7	286	21.4
	GA	120	283	185.67	—	75.73
P4S2	ABC	111	114	113.2	113	0.86
	DNLPSO	124	276	166.87	127	67.6
	GA (MATLAB)	114	118	115.67	116	1.23
	MPEDE	104	108	106.73	107	1.16
	TLBO	105	111	107.73	108	1.75
	GWO	108	–	111.9	112	2.2
	JAYA	110	–	116.2	116	2
	GA	105	117	109.96	—	1.94
P5S1	ABC	395	403	398.47	398	2.23
	DNLPSO	425	437	432.27	433	3.67
	GA (MATLAB)	398	414	408.87	409	3.48
	MPEDE	167	380	321.6	377	95.58
	TLBO	374	400	388.93	391	7.75
	GWO	167	–	352.2	380	73.8
	JAYA	392	–	399.4	400	3.6
	GA	165	389	349.26	—	75.43
P5S2	ABC	154	164	158.2	157	2.68
	DNLPSO	173	378	362.13	376	52.36
	GA (MATLAB)	161	170	165.53	165	2.47
	MPEDE	144	146	145.67	146	0.62
	TLBO	142	157	149.67	150	4.7
	GWO	151	–	157.8	158	3.5
	JAYA	158	–	164.2	164	2.8
	GA	146	167	151.63	—	3.60