Differential Evolution Algorithm for Permutation Flowshop Sequencing Problem with Makespan Criterion

M. Fatih Tasgetiren  
Dept. of Management, Fatih University  
34500 Buyukcekmece, Istanbul, Turkey  
Phone: 0212-889 08 10 ext 5024  
Email: ftasgetiren@fatih.edu.tr

Yun-Chia Liang  
Dept. of Industrial Engineering and Management  
Yuan Ze University  
No 135 Yuan-Tung Road, Chung-Li, Taoyuan County, Taiwan 320, R.O.C.  
Email: ycliang@saturn.yzu.edu.tw

Mehmet Sevkli  
Dept. of Industrial Engineering, Fatih University  
34500 Buyukcekmece, Istanbul, Turkey  
Phone: 0212-889 08 10 ext 1060  
Email: msevkli@fatih.edu.tr

Gunes Gencyilmaz  
Dept. of Management, Istanbul Kultur University  
E5 Karayolu Uzeri, Sirinevler, Istanbul, Turkey  
Email: g.gencyilmaz@iku.edu.tr
Differential Evolution Algorithm for Permutation Flowshop Sequencing Problem with Makespan Criterion

M. Fatih Tasgetiren\textsuperscript{a}, Yun-Chia Liang\textsuperscript{b}, Mehmet Sevkli\textsuperscript{c}, Gunes Gencyilmaz\textsuperscript{d}
\textsuperscript{a,c}Department of Management, Fatih University, 34500 Buyukcekmece, Istanbul, Turkey
\textsuperscript{b}Department of Industrial Engineering and Management, Yuan Ze University
\textsuperscript{No 135 Yuan-Tung Road, Chung-Li, Taoyuan County, Taiwan 320, R.O.C.}
\textsuperscript{d}Department of Management, Istanbul Kultur University, E5 Karayolu Uzeri, Sirinevler, Istanbul, Turkey

Abstract: This paper presents a differential evolution algorithm to solve the permutation flowshop sequencing problem with makespan criterion. Differential evolution is one of the latest evolutionary optimization algorithm applied to continuous optimization problems where members of population use chromosomes based on floating-point numbers to represent candidate solutions. In this paper we also present a heuristic rule, called smallest parameter value first in the permutation, which enables the differential evolution algorithm to be applied to all classes of sequencing scheduling problems. The results for the well known benchmark suite in the literature is presented and compared to the well known approaches such as genetic algorithm and particle swarm optimization algorithm.

Keywords: Genetic algorithm, Particle swarm optimization; Differential evolution, Permutation flowshop sequencing; Makespan
Differential Evolution Algorithm for Permutation Flowshop Sequencing Problem with Makespan Criterion

1. Introduction

Differential evolution (DE) is one of the latest evolutionary optimization methods proposed by Storn and Price [27]. Like the particle swarm optimization algorithms, candidate solutions are represented as chromosomes based on floating-point numbers. In DE, target population is perturbed with a mutant factor to establish a mutant population. Crossover operator is introduced to combine the mutated population with the target population so as to generate a trial population. Then selection operator is applied to compare the fitness function value of both competing populations, namely, target and trial population. Ultimately better individuals become members of the population for the next generation. This process is repeated until a convergence occurs.

Since DE was first introduced to optimize various continuous nonlinear functions by Storn and Price [27,28], it has been successfully applied in a variety of applications in Lampinen [11,12]. Although the applications of DE on combinatorial optimization problems are still limited [1], DE has its advantages such as simple concept, immediately accessible for practical applications, simple structure, ease of use, speed to get the solutions, and robustness. However, the continuous nature of the algorithm prohibits DE to apply to combinatorial optimization problems. To compensate this drawback, Tasgetiren et al. [32,33] presented the smallest position value (SPV) rule, borrowed from the random key representation of Bean [2], for the particle swarm optimization (PSO) algorithm, which is developed by Kennedy and Eberhard [4,9,10], to convert a continuous position vector to a job permutation. It has been successfully applied to the single machine total weighted tardiness problem (SMTWTP) and the permutation flowshop sequencing problem (PFSP). The SPV rule can still be used in DE since smallest position value can be replaced by smallest parameter value to convert the continuous parameter values to a job permutation. Following the successful applications above, this paper aims at employing DE in solving PFSP with makespan criterion.

Here we consider the PFSP. Given the processing times \( p_{jk} \) for job \( j \) on machine \( k \), and a job permutation \( \pi = \{\pi_1, \pi_2, \ldots, \pi_n\} \) where \( n \) jobs \( j = 1, 2, \ldots, n \) will be sequenced through \( m \) machines \( k = 1, 2, \ldots, m \) using the same permutation. For \( n/m/P/C_{\text{max}} \) problem, \( C(\pi_j, m) \) denotes the completion time of job \( \pi_j \) on machine \( m \). Given the job permutation \( \pi = \{\pi_1, \pi_2, \ldots, \pi_n\} \), the calculation of completion time for \( n \)-job \( m \)-machine problem is given as follows:

\[
C(\pi_j, l) = p_{\pi_j, l},
C(\pi_j, l+1) = C(\pi_{j-1}, l) + p_{\pi_{j-1}, l} \quad j = 2, \ldots, n
\]

\[
C(\pi_j, k) = C(\pi_{j-1}, k) + p_{\pi_{j-1}, k} \quad k = 2, \ldots, m
\]

Then makespan can be defined as

\[
C_{\text{max}}(\pi) = C(\pi_n, m)
\]

So, the PFSP with the makespan criterion is to find a permutation \( \pi^* \) in the set of all permutations \( \Pi \) such that

\[
C_{\text{max}}(\pi^*) \leq C(\pi_n, m) \quad \forall \pi \in \Pi.
\]

For the computational complexity of the PFSP with makespan criterion, Rinnooy Kan [24] proves that makespan minimization is NP-hard. Therefore, efforts have been devoted to finding high-quality solutions in a reasonable computational time by heuristic optimization techniques instead of finding optimal solution. Heuristics for the makespan minimization problem have been proposed in [3,5,6,8,13,14,18,19,29]. To achieve a better solution quality, modern meta-heuristics have been presented for the PFSP with makespan minimization such as Simulated Annealing in [16,17], Tabu
Search in [7,15,21,31], Genetic Algorithms in [22,23], Ant Colony Optimization in [20,26], and iterated local search in [25], Particle Swarm Optimization in [33]. This paper is organized as follows. Section 2 gives the methodology of the proposed DE algorithm, and computational results of test problems are shown in section 3. Finally, section 4 summarizes the concluding remarks.

2. PSO Algorithm for PFSP

Currently, there are several variants of DE. We follow the \textit{DE/rand/1/bin} scheme of Storn and Price [27] with the inclusion of SPV rule of Tasgetiren et al. [32, 33] in the algorithm. Pseudo code of the DE algorithm for the PFSP is given in Figure 1.

\textit{Initialize parameters}
\textit{Initialize target population}
\textit{Find permutation}
\textit{Evaluate}
\textit{Generation=0}
\textbf{Do} { 
\begin{align*}
& \text{Generation++} \\
& \text{Obtain the mutant population} \\
& \text{Obtain the trial population} \\
& \text{Find permutation} \\
& \text{Evaluate trial population} \\
& \text{Selection} \\
& \text{Apply local search or mutation \ (optional)} \\
\end{align*}
\textbf{While} (Termination)
\textbf{Figure 1. DE Algorithm with Local Search for PFSP.}

The basic elements of DE algorithm is summarized as follows:

\begin{itemize}
  \item **Target individual**: $X^t_i$ denotes the $i^{th}$ individual in the population at generation $t$ and is represented by $n$ number of dimensions as $X^t_i = [x^t_{i1}, x^t_{i2}, \ldots, x^t_{in}]$, where $x^t_{ij}$ is the optimized parameter value of the $i^{th}$ individual with respect to the $j^{th}$ dimension ($j = 1,2,\ldots,n$).
  \item **Mutant individual**: $V^t_i$ denotes the $i^{th}$ individual in the population at generation $t$ and is represented by $n$ number of dimensions as $V^t_i = [v^t_{i1}, v^t_{i2}, \ldots, v^t_{in}]$, where $v^t_{ij}$ is the optimized parameter value of the $i^{th}$ individual with respect to the $j^{th}$ dimension ($j = 1,2,\ldots,n$).
  \item **Trial individual**: $U^t_i$ denotes the $i^{th}$ individual in the population at generation $t$ and is represented by $n$ number of dimensions as $U^t_i = [u^t_{i1}, u^t_{i2}, \ldots, u^t_{in}]$, where $u^t_{ij}$ is the optimized parameter value of the $i^{th}$ individual with respect to the $j^{th}$ dimension ($j = 1,2,\ldots,n$).
  \item **Target population**: $P^t$ is the set of $NP$ individuals in the population at generation $t$, i.e., $P^t = [X^t_1, X^t_2, \ldots, X^t_{NP}]$.
  \item **Mutant population**: $V^t$ is the set of $NP$ individuals in the population at generation $t$, i.e., $V^t = [V^t_1, V^t_2, \ldots, V^t_{NP}]$.
  \item **Trial population**: $U^t$ is the set of $NP$ individuals in the population at generation $t$, i.e., $U^t = [U^t_1, U^t_2, \ldots, U^t_{NP}]$.
  \item **Permutation**: We introduce a new variable $\pi^t_i$, which is a permutation of jobs implied by the individual $X^t_i$. It can be described as $\pi^t_i = [\pi^t_{ij1}, \pi^t_{ij2}, \ldots, \pi^t_{ijn}]$, where $\pi^t_{ij}$ is the assignment of job $j$ of the individual $i$ in the permutation at generation $t$.
  \item **Mutant constant**: $F \in (0,2)$ is a real constant which affects the differential variation between two individuals.
  \item **Crossover constant**: $CR \in (0,1)$ is a crossover constant which affects the diversity of population for the next generation.
\end{itemize}
Fitness function: In a minimization problem, the objective function is $f(\pi'_i \leftarrow X'_i)$ where $\pi'_i$ is the corresponding permutation of individual $X'_i$.

Termination criterion: It is a condition that the search process will be terminated. It might be a maximum number of generation or maximum CPU time to terminate the search.

2.1 Solution Representation

When designing the DE algorithm, a direct relationship between problem domain and DE individual lies on its solution representation. In order to construct a direct relationship between the problem domain and DE individual, we present $n$ number of dimensions for $n$ number of jobs. In other words, each dimension represents a typical job. In addition, the individual $X'_i = [x'_i1, x'_i2, ..., x'_in]$ corresponds to the continuous parameter values for $n$ number of jobs in the PFSP. The individual does not present a permutation. Instead, we use the SPV rule to determine the permutation implied by the parameter values $x'_i_j$ of target individual $X'_i$. Table 1 illustrates the solution representation of target individual $X'_i$ for the DE algorithm with its corresponding permutation. According to the SPV rule, the smallest parameter value is $x'_i_j = -1.20$, so the dimension $j=5$ is assigned to be the first job $\pi'_i_5$ in the permutation $\pi'_i$; the second smallest parameter value is $x'_i_j = -0.99$, so the dimension $j=2$ is assigned to be the second job $\pi'_i_2$ in the permutation $\pi'_i$, and so on. In other words, dimensions are sorted according to the SPV rule, i.e., according to the parameter values $x'_i_j$ to construct the permutation $\pi'_i$.

<table>
<thead>
<tr>
<th>Dimension, $j$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x'_i_j$</td>
<td>1.80</td>
<td>-0.99</td>
<td>3.01</td>
<td>-0.72</td>
<td>-1.20</td>
<td>2.15</td>
</tr>
<tr>
<td>Jobs, $\pi'_i$</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

This representation is unique in terms of finding new solutions since parameters of each individual are updated at each generation $t$ in the DE algorithm, thus resulting in different permutations at each generation $t$.

2.2 Initial Population

A population of individuals is constructed randomly for the DE algorithm of the PFSP. The continuous parameter values are established randomly. The following formula is used to construct the initial continuous parameter values of the individual uniformly:

$$x'_i_j = x_{\text{min}} + (x_{\text{max}} - x_{\text{min}}) \times r_i$$

where $x_{\text{min}} = -1.0, x_{\text{max}} = 1.0$, and $r_i$ is a uniform random number between 0 and 1. During the reproduction of the DE algorithm, it is possible to extend the search outside of the initial range of the search space. For this reason, parameter values violating the initial range are restricted to the feasible range as follows:

$$x'_i_j = x_{\text{min}} + (x_{\text{max}} - x_{\text{min}}) \times r_i$$

Population size is twice the number of dimensions. As the formulation of the PFSP suggests that the objective is to minimize the makespan, the fitness function value is the makespan for the individual $i$ in the population. That is,

$$f'_i(\pi'_i \leftarrow X'_i) = \left. C_{\text{max}} (\pi'_i, m) \right|_{m}$$

The complete computational procedure of the DE algorithm for the PFSP can be summarized as follows:

**Step 1: Initialization**
- Set $t=0$, $NP =$ twice the number of dimensions.
Generate $NP$ individuals randomly as explained before, $\{X_i^0, i = 1,2,\ldots, NP\}$ where $X_i^0 = [x_{i,1}^0, x_{i,2}^0, \ldots, x_{in}^0]$. Apply the SPV rule to find the permutation $\pi_i^0 = [\pi_{i,1}^0, \pi_{i,2}^0, \ldots, \pi_{in}^0]$ of individual $X_i^0$ for $i = 1,2,\ldots, NP$. Evaluate each individual $i$ in the population using the objective function $f_i^0(\pi_i^0 \leftarrow X_i^0)$ for $i = 1,2,\ldots, NP$.

**Step 2: Update generation counter**

$t = t + 1$

**Step 3: Generate mutant population**

For each target individual, $X_i^t$, $i = 1,2,\ldots, NP$, at generation $t$, a mutant individual, $V_i^{t+1} = [v_{i,1}^{t+1}, v_{i,2}^{t+1}, \ldots, v_{in}^{t+1}]$, is determined such that:

$$V_i^{t+1} = X_i^t + F(X_{b_i}^t - X_{c_i}^t)$$

where $a_i$ is the best individual so far in the population and $b_i$ and $c_i$ are two randomly chosen individuals from the population such that $(a_i \neq b_i \neq c_i)$. $F > 0$ is a mutant factor which affects the differential variation between two individuals.

**Step 4: Generate trial population**

Following the mutation phase, the crossover (recombination) operator is applied to obtain the trial population. For each mutant individual, $V_i^{t+1} = [v_{i,1}^{t+1}, v_{i,2}^{t+1}, \ldots, v_{in}^{t+1}]$, an integer random number between 1 and $n$, i.e., $D_i \in \{1,2,\ldots,n\}$, is chosen, and a trial individual, $U_i^{t+1} = [U_{i,1}^{t+1}, U_{i,2}^{t+1}, \ldots, U_{in}^{t+1}]$ is generated such that:

$$U_{ij}^{t+1} = \begin{cases} v_{ij}^{t+1}, & \text{if } r_{ij}^{t+1} \leq CR \text{ or } j = D_i \\ x_{ij}^t, & \text{Otherwise} \end{cases}$$

where the index $D$ refers to a randomly chosen dimension ($j=1,2,\ldots,n$), which is used to ensure that at least one parameter of each trial individual $U_i^{t+1}$ differs from its counterpart in the previous generation $U_i^t$. CR is a user-defined crossover constant in the range $[0, 1]$, and $r_{ij}^{t+1}$ is a uniform random number between 0 and 1. In other words, the trial individual is made up with some parameters of mutant individual, or at least one of the parameters randomly selected, and some other parameters of target individual.

**Step 5: Find Permutation**

Apply the SPV rule to find the permutation $\pi_i^{t+1} = [\pi_{i,1}^{t+1}, \pi_{i,2}^{t+1}, \ldots, \pi_{in}^{t+1}]$ for $i = 1,2,\ldots, NP$.

**Step 6: Selection**

Evaluate the trial population using the objective function $f_i^{t+1}(\pi_i^{t+1} \leftarrow U_i^{t+1})$ for $i = 1,2,\ldots, NP$.

**Step 7: Selection**

To decide whether or not the trial individual $U_i^{t+1}$ should be a member of the target population for the next generation, it is compared to its counterpart target individual $X_i^t$ at the previous generation. The selection is based on the survival of fitness among the trial population and target population such that:

$$X_i^{t+1} = \begin{cases} U_i^{t+1}, & \text{if } f(\pi_i^{t+1} \leftarrow U_i^{t+1}) \leq f(\pi_i^{t} \leftarrow X_i^t) \\ X_i^t, & \text{otherwise} \end{cases}$$

**Step 8: Stopping criterion**

If the number of generation exceeds the maximum number of generation, or maximum CPU time, then stop; otherwise go to step 2.

### 2.3 Neighborhood of DE

In a DE algorithm, two types of the local search are possible to be employed: local search applied to the parameters and local search applied to the permutations. The first one is based on the neighbors of the parameter values $x_j^t$ whereas the second one is based on the neighbors of jobs $\pi_j^t$ in the permutation $\pi_i^t$. As an example for the insert operator, removing the position value $x_{i,2}^t = -0.99$ from the
second dimension and inserting it into the fourth dimension results in a solution which is illustrated in Table 2.

<table>
<thead>
<tr>
<th>Dimensions, $j$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x'_{ij}$</td>
<td>1.80</td>
<td><strong>-0.99</strong></td>
<td>3.01</td>
<td>-0.72</td>
<td>-1.20</td>
<td>2.15</td>
</tr>
<tr>
<td>Jobs, $\pi'_{ij}$</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>$x'_{ij}$</td>
<td>1.80</td>
<td>3.01</td>
<td>-0.72</td>
<td><strong>-0.99</strong></td>
<td>-1.20</td>
<td>2.15</td>
</tr>
<tr>
<td>Jobs, $\pi'_{ij}$</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 2. Local Search Applied to Parameters

Note that this neighborhood search requires finding the permutation again implied by the individual’s parameter values explained in solution representation in Table 1 before its evaluation. In other words, once the position is inserted, the SPV rule should be applied to determine the permutation again. Instead, the second approach is very common and applicable to the permutation directly. However, it violates the SPV first rule and needs a repair algorithm. This approach is illustrated in Table 3 and 4, where job $\pi'_{i2} = 2$ and job $\pi'_{i4} = 1$ are interchanged.

<table>
<thead>
<tr>
<th>Dimensions, $j$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x'_{ij}$</td>
<td>1.80</td>
<td><strong>-0.99</strong></td>
<td>3.01</td>
<td>-0.72</td>
<td>-1.20</td>
<td>2.15</td>
</tr>
<tr>
<td>Jobs, $\pi'_{ij}$</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>$x'_{ij}$</td>
<td>1.80</td>
<td>3.01</td>
<td>-0.72</td>
<td><strong>-0.99</strong></td>
<td>-1.20</td>
<td>2.15</td>
</tr>
<tr>
<td>Jobs, $\pi'_{ij}$</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3. Local Search Applied to Permutation Before Repairing

As seen in Table 3, applying a local search to the permutation violates the SPV rule because the permutation itself is a result of the individual’s parameter values. Once a local search is completed, individual should be repaired so that the SPV is not violated. This is achieved by changing the parameter values according to the SPV rule as shown in Table 4.

<table>
<thead>
<tr>
<th>Dimensions, $j$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x'_{ij}$</td>
<td><strong>-0.99</strong></td>
<td>1.80</td>
<td>3.01</td>
<td>-0.72</td>
<td>-1.20</td>
<td>2.15</td>
</tr>
<tr>
<td>Jobs, $\pi'_{ij}$</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4. Local Search Applied to Permutation After Repairing

In other words, interchange the position values of the exchanged jobs in terms of their dimensions. Since jobs $\pi'_{i2} = 2$ and $\pi'_{i4} = 1$ are interchanged, their associated parameter values, $x'_{i2} = -0.99$ and $x'_{i4} = 1.80$, are interchanged for dimensions $j=2$ and $j=1$ to keep the individual consistent with the SPV rule. The advantage of this method is due to the fact that the repair algorithm is only needed after evaluating all the neighbors in a permutation. The second approach is employed in this paper.

3. Experimental Results

The proposed DE$_{SPV}$ algorithm for the PFSP is coded in C and run on an Intel P4 2.6 GHz PC with 256MB memory. The traditional GA and PSO algorithm has been presented in Tasgetiren et al. [33]. In GA developed, permutation representation was used, population size was twice the number of jobs, crossover and mutation probabilities were taken as 1.0 and 0.05 respectively. For the mate selection, one individual was selected randomly and the other one by the tournament selection with size of 2.
Again the tournament selection with size of 2 was employed to construct the population for the next generation. In line with the PSO algorithm, GA also employed the insert operator as a mutation scheme. Regarding the PSO parameters, social and cognitive parameters were taken as $c_s = c_c = 2$ consistent with the literature. Initial inertia weight was set to $w^0 = 0.9$ and was never decreased below 0.40. Finally, the decrement factor $\beta$ was taken as 0.975. Population size was twice the number of dimensions (jobs). The performance of GA and PSO was evaluated by using the benchmark suite of Taillard [30], which can be found in [http://ina.eivd.ch/collaborateurs/etd/default.htm](http://ina.eivd.ch/collaborateurs/etd/default.htm). Regarding the DE parameters, $F$ and $CR$ are taken as 0.4 and 0.5 respectively. In order to show the impact of the local search on the solution quality, the DE algorithm employs the exchange operator as a local search with the size of number of jobs. In other words, instead of employing the insert operator as a mutation scheme, simple exchange operator is applied to the best-sofar solution of the DE algorithm at each iteration in a way that two randomly chosen jobs in the permutation of the best-sofar solution are exchanged. Size of this local search is the number of jobs. DE with the local search is denoted as $DE_{spv+exchange}$.

The solution quality is measured with the percent relative increase in makespan with respect to the upper bounds provided by Taillard [30]. To be more specific, $\Delta_{avg}$ is computed as follows:

$$\Delta_{avg} = \frac{100}{R} \sum_{i=1}^{R} \left( \frac{H_i - U_i}{U_i} \right)$$

where $H_i$ denotes the value of the makespan that GA, PSO$_{spv}$ or DE$_{spv}$ algorithms generated whereas $U_i$ is the value of upper bounds for makespan provided by Taillard [30], and $R$ is the total number of replications, i.e., $R$ is equal to the number of replications for each instance times the total number of instances. For the computational effort consideration, $t_{avg}$ denotes the average CPU times over $R$ runs in seconds.

For the makespan criterion, we run each instance for 500 generations and 10 replications are carried out for each instance to obtain the statistics for $DE_{spv}$ and $DE_{spv+exchange}$ algorithms as in Tasgetiren et al.[33]. Table 5 shows the results with respect to makespan minimization for the Taillard’s benchmark suite. In terms of the mean percent relative increase in makespan ($\Delta_{avg}$), $DE_{spv}$ produced slightly better results than both GA and PSO$_{spv}$. However, we would like to point out the difference in the CPU times. Since PSO updates both velocity and position vector at each iteration, it consumes more CPU time than both GA and DE. For the small size of problems, CPU times are identical. However, as the problem size becomes larger, PSO algorithm suffers from updating all dimensions both for velocity and position vectors. In summary, three algorithms show similar performance in terms of relative percent increase in makespan, but PSO is more time consuming from both GA and DE. This observation is an advantage of DE algorithms over PSO algorithms. In addition, inclusion of a simple exchange operator as a local search in the $DE_{spv}$ algorithm has improved the results significantly since it is decreased from 3.53 to 1.78 at the expense of slightly increase in the CPU time.

Table 5. Performance Comparison on Taillard’s Benchmarks with respect to $C_{max}$

<table>
<thead>
<tr>
<th></th>
<th>$\Delta_{avg}$</th>
<th>$\Delta_{std}$</th>
<th>$t_{avg}$</th>
<th>$\Delta_{avg}$</th>
<th>$\Delta_{std}$</th>
<th>$t_{avg}$</th>
<th>$\Delta_{avg}$</th>
<th>$\Delta_{std}$</th>
<th>$t_{avg}$</th>
<th>$\Delta_{avg}$</th>
<th>$\Delta_{std}$</th>
<th>$t_{avg}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20x5</td>
<td>3.13</td>
<td>1.86</td>
<td>0.28</td>
<td>1.71</td>
<td>1.25</td>
<td>0.19</td>
<td>2.25</td>
<td>1.37</td>
<td>0.16</td>
<td>0.69</td>
<td>0.64</td>
<td>0.19</td>
</tr>
<tr>
<td>20x10</td>
<td>5.42</td>
<td>1.72</td>
<td>0.31</td>
<td>3.28</td>
<td>1.19</td>
<td>0.23</td>
<td>3.71</td>
<td>1.24</td>
<td>0.19</td>
<td>2.01</td>
<td>0.93</td>
<td>0.24</td>
</tr>
<tr>
<td>20x20</td>
<td>4.22</td>
<td>1.31</td>
<td>0.45</td>
<td>2.84</td>
<td>1.15</td>
<td>0.30</td>
<td>3.03</td>
<td>0.98</td>
<td>0.52</td>
<td>1.85</td>
<td>0.87</td>
<td>0.34</td>
</tr>
<tr>
<td>50x5</td>
<td>1.69</td>
<td>0.79</td>
<td>1.46</td>
<td>1.15</td>
<td>0.70</td>
<td>1.40</td>
<td>0.88</td>
<td>0.52</td>
<td>0.92</td>
<td>0.41</td>
<td>0.37</td>
<td>1.17</td>
</tr>
<tr>
<td>50x10</td>
<td>5.61</td>
<td>1.41</td>
<td>1.55</td>
<td>4.83</td>
<td>1.16</td>
<td>1.58</td>
<td>4.12</td>
<td>1.10</td>
<td>1.17</td>
<td>2.41</td>
<td>0.90</td>
<td>1.45</td>
</tr>
<tr>
<td>50x20</td>
<td>6.95</td>
<td>1.09</td>
<td>2.03</td>
<td>6.68</td>
<td>1.35</td>
<td>2.10</td>
<td>5.56</td>
<td>1.22</td>
<td>1.63</td>
<td>3.59</td>
<td>0.78</td>
<td>2.08</td>
</tr>
<tr>
<td>100x5</td>
<td>0.81</td>
<td>0.39</td>
<td>6.54</td>
<td>0.59</td>
<td>0.34</td>
<td>10.39</td>
<td>0.44</td>
<td>0.29</td>
<td>5.43</td>
<td>0.21</td>
<td>0.21</td>
<td>6.69</td>
</tr>
<tr>
<td>100x10</td>
<td>3.12</td>
<td>0.95</td>
<td>6.88</td>
<td>3.26</td>
<td>1.04</td>
<td>10.87</td>
<td>2.28</td>
<td>0.75</td>
<td>6.36</td>
<td>1.41</td>
<td>0.57</td>
<td>7.43</td>
</tr>
<tr>
<td>100x20</td>
<td>6.32</td>
<td>0.89</td>
<td>8.57</td>
<td>7.19</td>
<td>0.99</td>
<td>12.76</td>
<td>6.78</td>
<td>1.12</td>
<td>8.10</td>
<td>3.11</td>
<td>0.55</td>
<td>9.90</td>
</tr>
<tr>
<td>200x10</td>
<td>2.08</td>
<td>0.45</td>
<td>32.13</td>
<td>2.47</td>
<td>0.71</td>
<td>81.81</td>
<td>1.88</td>
<td>0.69</td>
<td>38.89</td>
<td>1.06</td>
<td>0.35</td>
<td>45.08</td>
</tr>
<tr>
<td>200x20</td>
<td>5.10</td>
<td>0.60</td>
<td>36.88</td>
<td>7.08</td>
<td>0.81</td>
<td>89.77</td>
<td>7.94</td>
<td>0.75</td>
<td>45.48</td>
<td>2.81</td>
<td>0.47</td>
<td>51.38</td>
</tr>
<tr>
<td>Mean</td>
<td>4.04</td>
<td>1.04</td>
<td>8.83</td>
<td>3.73</td>
<td>0.97</td>
<td>19.22</td>
<td>3.53</td>
<td>0.91</td>
<td>9.90</td>
<td>1.78</td>
<td>0.61</td>
<td>11.45</td>
</tr>
</tbody>
</table>
4. Conclusions

To the best of our knowledge, this is the first reported application of the DE algorithm to the Taillard’s benchmark suite for PFSP with makespan criterion. We show that the SPV rule presented in Tasgetiren et al. [32,33] can also be used in DE algorithms to convert a continuous parameter vector to a job permutation so that DE can be used for solving all classes of combinatorial optimization problems. We also show that PSO algorithms may computationally be very expensive as the problem size becomes larger.

In summary, the results presented in this paper is very encouraging and promising for application of DE algorithms to the combinatorial optimization problems. For a future research, further analysis should be required to see the impact of population size, and maximum number of generations (iterations) on the solution quality along with the CPU time requirements.

References


