A Discrete Differential Evolution Algorithm for the Permutation Flowshop Scheduling Problem

Quan-Ke Pan
College of Computer Science, Liaocheng University, Liaocheng, Shandong Province, 252059, P. R. China
qkpan@lcu.edu.cn

M. Fatih Tasgetiren
Department of Operations Management and Business Statistics, Sultan Qaboos University, Muscat, Oman
mfatih@squ.edu.om

Yun-Chia Liang
Department of Industrial Engineering and Management, Yuan Ze University, No 135 Yuan-Tung Road, Chung-Li, Taoyuan County, Taiwan
ycliang@saturn.yzu.edu.tw

ABSTRACT

In this paper, a novel discrete differential evolution (DDE) algorithm is presented to solve the permutation flowhop scheduling problem with the makespan criterion. The DDE algorithm is simple in nature such that it first mutates a target population to produce the mutant population. Then the target population is recombined with the mutant population in order to generate a trial population. Finally, a selection operator is applied to both target and trial populations to determine who will survive for the next generation based on fitness evaluations. As a mutation operator in the discrete differential evolution algorithm, a destruction and construction procedure is employed to generate the mutant population. We propose a referenced local search, which is embedded in the discrete differential evolution algorithm to further improve the solution quality. Computational results show that the proposed DDE algorithm with the referenced local search is very competitive to the iterated greedy algorithm which is one of the best performing algorithms for the permutation flowshop scheduling problem in the literature.

Categories and Subject Descriptors

I.2.8 [**Computing Methodology**]: Problem Solving, Control Methods, and Search – *heuristic methods, scheduling*

General Terms

Algorithms

Keywords

Scheduling; Particle swarm optimization; Permutation flowshop; Makespan; Discrete differential evolution.

1. INTRODUCTION

The Permutation Flowshop Sequencing Problem (PFSP) basically deals with finding a permutation of jobs on machines such that certain performance measures will be minimized and the same job permutation applies to each machine. Flowshop problems have attracted the attention of researchers since the proposal of the problem by Johnson [1]. Among the practical performance measures, the minimization of makespan are known to lead to the minimization of total production run, stable utilization of resources, rapid turn-around of jobs, and the minimization of work-in-process (WIP) inventory.

The formulation of the PFSP can be given as follows: Given the processing times p_{jk} for job j and machine k, and a job permutation $\pi = \{\pi_1, \pi_2, ..., \pi_n\}$ where n jobs $(j = 1,2,...,n)$ will be sequenced through m machines $(k = 1,2,...,m)$, then the problem is to find the best permutation of jobs to be valid for each machine. For $n/m/P/C_{\max}$ problem, $C(\pi_j, m)$ denotes the completion time of the job π_j on the machine m. Given the job permutation $\pi = \{\pi_1, \pi_2, ..., \pi_n\}$, the calculation of completion time for the n-job, m-machine problem is given as follows:

$$C(\pi_1,1) = p_{\pi_1,1}$$

$$C(\pi_j,1) = C(\pi_{j-1},1) + p_{\pi_j,1} \qquad j = 2,...,n$$

$$C(\pi_1,k) = C(\pi_1,k-1) + p_{\pi_1,k} \qquad k = 2,...,m$$

$$C(\pi_j,k) = \max\{C(\pi_{j-1},k), C(\pi_j,k-1)\} + p_{\pi_j,k} \quad j = 2,...,n; k = 2,...,m$$

Then makespan can be defined as

$$C_{\max}(\pi) = C(\pi_n, m). \qquad (1)$$

So, the PFSP with the makespan criterion is to find a permutation π^* in the set of all permutations $\prod$ such that

$$C_{\max}(\pi^*) \le C(\pi_n, m) \quad \forall \pi \in \prod.$$

For the computational complexity of the PFSP with makespan objectives, Rinnooy Kan [2] proved to be NP-complete. Therefore, efforts have been devoted to finding high-quality or near-optimal solutions in a reasonable computational time by heuristic optimization techniques instead of finding an optimal solution. Heuristics for the makespan minimization problem have been proposed by Palmer [3], Campbell et al. [4], Dannenbring [5], Nawaz et al. [6], Taillard [7], Framinan et al. [8] and

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

Framinan and Leisten [9]. To achieve a better solution quality, modern meta-heuristics have been presented for the PFSP with makespan minimization such as Ant Colony Optimization in [10, 11], Genetic Algorithm in [12, 13, 14], Iterated Local Search in [15], Simulated Annealing in [16, 17], Tabu Search in [18, 19, 20]. Iterated Greedy Algorithm in [21]. An excellent review of flowshop heuristics and metaheuristics can be found in [22]. In order to test the performance of these heuristics, the 120 benchmark instances presented by Taillard [23] are generally used in these modern heuristic algorithms.

Differential evolution (DE) is one of the latest evolutionary optimization methods proposed by Storn & Price [24]. Like other evolutionary-type algorithms, DE is a population-based and stochastic global optimizer. In a DE algorithm, candidate solutions are represented by chromosomes based on floating-point numbers. In the mutation process of a DE algorithm, the weighted difference between two randomly selected population members is added to a third member to generate a mutated solution. Then, a crossover operator follows to combine the mutated solution with the target solution so as to generate a trial solution. Thereafter, a selection operator is applied to compare the fitness function value of both competing solutions, namely, target and trial solutions to determine who can survive for the next generation. Since DE was first introduced to solve the Chebychev polynomial fitting problem by Storn & Price [24], it has been successfully applied in a variety of applications that can be found in Price et al. [25] and Babu & Onwubolu [26]. Regarding the applications of differential evolution algorithm to scheduling problems, related literature can be found in [32, 33, 34, 35].

The applications of DE on combinatorial optimization problems are still limited, but the past experience of successfully applying DE algorithms to combinatorial problems in the literature [27] has proved the promising of DE on some scheduling problems. For this reason, this research presents a discrete differential evolution (DDE) algorithm to solve the permutation flowshop scheduling problem with the makespan criterion.

The remaining paper is organized as follows. Section 2 introduces the discrete differential evolution (DDE) algorithm. Computational results are discussed in Section 3. Finally, Section 4 summarizes the concluding remarks.

2. DDE ALGORITHM

Currently, there exist several mutation variations of DE. The *DE/rand/1/bin* scheme of Storn & Price [24] is presented below. The DE algorithm starts with initializing the initial target population $\pi_i = [\pi_1, \pi_2, .., \pi_{NP}]$ with the size of *NP*. Each individual has an *n*-dimentional vector with parameter values determined randomly and uniformly between predefined search range. To generate a mutant individual, DE mutates vectors from the target population by adding the weighted difference between two randomly selected target population members to a third member at iteration *t* as follows:

$$v_{ij}^t = \pi_{aj}^{t-1} + F\left(\pi_{bj}^{t-1} - \pi_{cj}^{t-1}\right) \quad (2)$$

where a, b, and c are three randomly chosen individuals from the target population such that $(a \neq b \neq c \in (1,..,NP))$ and $j = 1,..,n$. $F > 0$ is a mutation scale factor which affects the differential variation between two individuals. Following the mutation phase, the crossover operator is applied to obtain the trial individual such that:

$$u_{ij}^t = \begin{cases} v_{ij}^t & if \quad r_{ij}^t \le CR \quad or \quad j = D_j \\ \pi_{ij}^{t-1} & otherwise \end{cases} \quad (3)$$

where the D_j refers to a randomly chosen dimension $(j = 1,..,n)$, which is used to ensure that at least one parameter of each trial individual u_{ij}^t differs from its counterpart in the previous generation u_{ij}^{t-1}. CR is a user-defined crossover constant in the range [0, 1], and r_{ij}^t 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 the target individual.

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

$$\pi_i^t = \begin{cases} u_i^t & if \quad f\left(u_i^t\right) \le f\left(\pi_i^{t-1}\right) \\ \pi_i^{t-1} & otherwise \end{cases} \quad (4)$$

Again note that standard DE equations cannot be used to generate discrete/binary values since positions are real-valued. Instead we propose a new and novel DDE algorithm whose solutions are based on discrete/binary values, which can be applied to all types of combinatorial optimization problems. In the DDE algorithm, the target population is constructed based on permutation of jobs as represented by $\pi_i = [\pi_1, \pi_2, .., \pi_{NP}]$. For the mutant population the following equations can be used:

$$V_i^t = P_m \oplus F_k\left(\pi_i^{t-1}\right) \quad (5)$$

$$V_i^t = P_m \oplus F_k\left(\pi_a^{t-1}\right) \quad (6)$$

$$V_i^t = P_m \oplus F_k\left(\pi_g^{t-1}\right) \quad (7)$$

Where π_i^{t-1} is the i*th* individual from the target population at iteration *t*-1; π_a^{t-1} is a randomly chosen individual from the target population at iteration *t*-1; π_g^{t-1} is the global best solution at iteration *t*-1; P_m is the mutation probability; and F_k is the mutation operator with the mutation strength of *k*. Suppose that the equation (7) is employed as a mutation operator. A uniform random number *r* is generated between [0, 1]. If *r* is less than P_m then the mutation operator is applied to generate the mutant individual $V_i^t = F_k\left(\pi_g^{t-1}\right)$ at current iteration *t*, otherwise the global best solution is kept as $V_i^t = \pi_g^{t-1}$. In the mutation equation, *k* represents the mutation strength. The lower the value of mutation strength *k* is, the lower the possibility that the algorithm would avoid getting stuck at the local minima. On the

other hand, the higher the value of mutation strength k is, the higher the possibility that the algorithm would possess excessive randomness. So care must be taken in the choice of the value of the mutation strength. It should be noted that we employed the destruction and construction procedure of the iterated greedy algorithms in the mutation phase of the DDE algorithm. Following the mutation phase, the trial individual is obtained such that:

$$U_i^t = P_c \oplus CR\left(\pi_i^{t-1}, V_i^t\right) \tag{8}$$

where CR is the crossover operator, and P_c is the crossover probability. In other words, the *ith* individual is recombined with its corresponding mutant individual using the crossover operator CR to generate the trial individual if a uniform random number r is less than the crossover probability P_c , then the crossover operator is applied to generate the trial individual $U_i^t = CR\left(\pi_i^{t-1}, V_i^t\right)$. Otherwise the trial individual is chosen as $U_i^t = V_i^t$. By doing so, the trial individual is made up either from the outcome of mutation operator or the crossover operator.

Finally, the selection is based on the survival of the fitness among the trial individual at the current iteration t and target individual at the previous iteration t-1 such that:

$$\pi_i^t = \begin{cases} U_i^t & \text{if } f\left(U_i^t\right) \le f\left(\pi_i^{t-1}\right) \\ \pi_i^{t-1} & \text{otherwise} \end{cases} \tag{9}$$

2.1 Solution Representation

In order to handle the PFSP properly, particles are represented by the permutation of jobs in n dimensions. Solution representation is given in Figure 1 where π_{ij} denotes the *jth* dimension/job of the ith particle.

j	1	2	3	.	.	n
π_{ij}	π_{i1}	π_{i2}	π_{i3}	.	.	π_{in}

Figure.1. Solution Representation.

Then, the fitness function of the particle is the makespan and given by

$$F_i = C_{\max}\left(\pi_i\right) = C\left(\pi_n, m\right). \tag{10}$$

For simplicity, we omit the index i of particle π_i from the representation from now on.

2.2 NEH Heuristic

The NEH heuristic of Nawaz et al. [6] has two phases which can be explained as follows:

1. In the first phase, jobs are ordered in descending sums of their processing times such that

$$P_j = \sum_{k=1}^{m} p_{jk}\ , \quad j = 1,..,n$$

2. In the second phase, the first two jobs are chosen so that their two possible sequences will be evaluated to establish the partial schedule. Next, a job permutation is established by evaluating the partial schedules based on the initial order of the first phase. Suppose a current permutation is already determined for the first π jobs, $\pi+1$ partial permutations are constructed by inserting job $\pi+1$ in $\pi+1$ possible slots of the current permutation. Among these $\pi+1$ permutations, the best one generating the minimum makespan is kept as the current permutation for the next iteration. Then job $\pi+2$ from the first phase is considered and so on until all jobs have been sequenced.

The computational complexity of the NEH heuristic is $O\left(n^3 m\right)$, which can consume considerable CPU time for large instances. However, Taillard [7] introduced a speed-up method which reduces the complexity of NEH to $O\left(n^2 m\right)$. This speed-up method is one of the key factors to success of most algorithms presented for permutation flowshop scheduling problem in the literature. For this reason, we also employ it in our any implementation of the NEH heuristic as well as in the construction phase of the IG algorithm embedded in the DDE algorithm proposed.

2.3 Iterated Greedy Algorithm

Iterated greedy (IG) algorithm has been successfully applied to the Set Covering problem (SCP) in Jacobs and Brusco [28], and Marchiory and Steenbeek [29], and the permutation flowshop scheduling problem in Ruiz and Stützle [21]. In an IG algorithm, solutions are simply generated in an iterated greedy (IG) algorithm using the main idea of destruction and construction. Destruction phase is concerned with removing some solution components from a previously constructed solution whereas construction phase is related to the reconstruction of a complete solution by using a greedy heuristic. An acceptance criterion is then used to decide whether or not the reconstructed solution will replace the incumbent solution. These simple steps are iterated until a predetermined termination criterion is met [21].

The key procedures in any IG algorithm are the destruction and construction phases applied to the DDE individual. d jobs from the individual are chosen randomly to be removed so that the partial permutation of the particle with $n-d$ jobs will be established, which is denoted as π_D as well as the set of d jobs, which is denoted as π_R to be reinserted onto π_D . The construction phase requires a heuristic procedure to reinsert the π_R jobs in a greedy manner. In other words, the first job in the set π_R is reinserted into all possible $n-d+1$ slots in the partial permutation π_D . Among these $n-d+1$ insertions, the best one with minimum makespan is chosen as the current partial permutation for the next insertion. Then the second job in the set π_R is considered and so on until π_R is empty. The destruction and construction procedure is illustrated in the following example with 5 jobs with the destruction size of d=2.

CURRENT PERMUTATION

j	1	2	3	4	5
π_j	3	1	4	5	2

DESTRUCTION PHASE

Step 1.a. Choose $d = 2$ jobs, randomly.

j	1	**2**	3	**4**	5
π_j	3	**1**	4	**5**	2

Step 1.b. Establish $\pi_D = \{3,4,2\}$, $\pi_R = \{1,5\}$.

d	1	2	3	r	1	2
π_D	3	4	2	π_R	1	5

CONSTRUCTION PHASE

Step2.a. After the best insertion of the job $\pi_{R,1} = 1$ in 3+1 possible slots.

d	1	2	3	4	r	2
π_D	3	4	**1**	2	π_R	5

Step2.b. After the best insertion of the job $\pi_{R,2} = 5$ in 4+1 possible slots.

j	1	2	3	4	5
π_j	**5**	3	4	1	2

2.4 Two-Cut PTL Crossover

Two-cut PTL crossover operator presented in [30] is used to update the particles of the DPSO algorithm. Two-cut PTL crossover operator is able to produce a pair of distinct offspring even from two identical parents. An illustration of two-cut PTL crossover operator is shown in Figure 2.

Two-Cut PTL Crossover						Two-Cut PTL Crossover					
P1	5	**1**	**4**	2	3	P1	5	**1**	**4**	2	3
P2	*3*	*5*	**4**	*2*	**1**	P2	*5*	**1**	**4**	*2*	*3*
O1	*3*	*5*	*2*	**1**	**4**	O1	*5*	*2*	*3*	**1**	**4**
O2	**1**	**4**	*3*	*5*	*2*	O2	**1**	**4**	*5*	*2*	*3*

Figure 2. An Example of the PTL Crossover Operator.

In the PTL crossover, a block of jobs from the first parent is determined by two cut points randomly. This block is either moved to the right or left corner of the permutation. Then the offspring permutation is filled out with the remaining jobs from the second parent. This procedure will always produce two distinctive offspring even from the same two parents as shown in Figure 2. In this paper, one of these two unique offspring is chosen randomly with an equal probability of 0.5.

2.5 Referenced Local Search

The local search what we call it referenced local search (RLS) is inspired from the job index based insertion scheme (JIBIS) of Rajendran [31]. Instead of using the job index of the current permutation, the RLS uses the reference permutation π^R taken from the search procedure such as the bestsofar solution, NEH solution, JIBIS solution or the best solution in the initial population. After constructing the initial population, we establish a sequence where the jobs are arranged in descending sum of their processing times. Then we apply the Referenced Insertion Scheme (RIS) to the global best solution of the initial population. The solution returned by the RIS is set to the reference permutation and used as a reference throughout the algorithm. The basic idea behind it is to take reference of positions of the jobs from a good permutation and insert them in different positions in the incumbent permutation to find a better permutation. It should be noted that we apply the local search to the global best solution π^t_g at each iteration t. The referenced local search and the referenced insertion procedure are shown in Figure 3 and 4, respectively.

Procedure RLS(π_g)

```
π:=DestructConstruct(π_g);
π_1:=RIS(π)
If C_max(π_1)<C_max(π) then
  π:=π_1;
Else
  If (random<exp(-(C_max(π)-C_max(π_1)/T))) then
    π:=π_1;
  Endif
Return π
End.
```

Fig. 3. Referenced Local Search.

Procedure RIS(π)

```
π* := π^R
Set h:=1;
Set i:=1;
while(i<n) do
  h:=(h mod n);
  Remove the job π_d from π, which corresponds to the job π*_h.
  π_1:=the best permutation obtained by inserting job π_d in any
   possible position of π.
  If C_max(π_1)< C_max(π) then
     π:=π_1;
     i:=1;
  else
     i:=i+1;
  endif
     h:=h+1;
  end while
  return π
end.
```

Figure 4. Referenced Insertion Scheme.

A constant temperature is used in the simulated annealing type of acceptance criterion in the DDE algorithm as suggested by Osman and Potts [17]:

$$T = \frac{\sum_{j=1}^{n}\sum_{k=1}^{m} p_{jk}}{n \times m \times 10} \times \tau \qquad (11)$$

where $\tau = 0.4$. In this way, the global best solution is diversified by giving chances to some inferior solutions during the search to escape from the local minima. The pseudo code of the DDE algorithm for the PFSP is given in Figure 5.

Procedure DDE

initialize parameters
initialize target population
evaluate target population
π^R=sequence that jobs are arranged in descending sum of their processing times
π^R :=**RIS**(π_g).
while (not termination) **do**
obtain mutant population
obtain trial population
evaluate trial population
make selection
apply local search **RLS**(π_g)
endwhile
return globalbest
end

Figure 5. DDE Algorithm for the PFSP.

The DDE algorithm with the referenced local search will be denoted as DDE_{RLS} from now on throughout the paper.

3. COMPUTATIONAL RESULTS

The basic objective of this study is to compare the performance of the DDE_{RLS} algorithm with the IG_RS_{LS} algorithm recently presented in Ruiz & Stutzle [21]. Even though we obtained the IG_RS_{LS} code through personal communication, we have developed our own IG_RS_{LS} code to run both algorithms in the same machine environment.

To give a brief and sound explanation about the DDE_{RLS} algorithm presented, the destruction and construction heuristic with destruction size of 4 (d=4) is used to generate the mutant population. In the construction phase, the NEH heuristic with the speed-up method of Taillard is utilized. No such effort has been devoted to adjusting the parameters of the DDE_{RLS} algorithm due to the following facts:

1. Ruiz & Stutzle [21] have already conducted a detailed design of experiments for parameter setting of the destruction size and the temperature parameter of the acceptance criterion. For these reasons, we just simply took the destruction size and temperature parameter as d=4 and τ=0.4, respectively as in Ruiz & Stutzle [21]. Two-cut PTL crossover is used in the update equation (15);
2. Regarding the other parameters of the DDE_{RLS} algorithms, population size is set to NP =20, mutation probability to P_m =0.2, and the crossover probability to P_c =0.8. The reason for which the low mutation and population size were taken was to give more chances to the RLS local search algorithm since it is well-known that the performance of evolutionary algorithms without a good local search is not satisfactory to solve the discrete optimization problems. However, we again show that the hybridization of an evolutionary algorithm with a good local search enhances its performance significantly.

DDE_{RLS} and IG_RS_{LS} algorithms for the PFSP problem were coded in Visual C++ and run on an Intel P IV 3.0 GHz PC with 512MB memory. Both algorithms were applied to the 120 benchmark instances of Taillard [23] ranging from 20 jobs with 5 machines to 500 jobs with 20 machines. Termination criterion is set to $n \times (m/2) \times t$ where t is taken as 30, 60 and 90 milliseconds as in Ruiz & Stutzle [21].

R=5 runs were conducted for each problem instance consistent with Ruiz & Stutzle [21]. The average relative percentage deviation (ARPD) and the average CPU time to the best makespan, i.e., the time that the makespan does not change after that point of time, in each replication averaged over R runs were given as statistics for performance measures. The average relative percentage deviation was computed as follows:

$$ARPD = \sum_{i=1}^{R} \left(\frac{(M_i - M_{REF}) \times 100}{M_{REF}} \right) / R \qquad (12)$$

where M_i was the makespan by the DDE_{RLS} or IG_RS_{LS} algorithms in each run whereas M_{REF} was the optimal or the lowest known upper bound for Taillard's instances as of April 2004, and R was the number of runs.

Computational results are given in Tables 1-6. We only compare the DDE_{RLS} algorithm to the IG_RS_{LS} algorithm since the IG_RS_{LS} algorithm has already been shown to be superior to 12 best performing algorithms compared in [21]. There exist several other sophisticated algorithms in the literature such as TSAB [18], RY [13], TSGW [20]. However, our main goal is to present the superior performance of both the IG_RS_{LS} and DDE_{RLS} algorithms.

As seen in Table 1-6, the performance of the IG_RS_{LS} algorithm was better than the results in Ruiz and Stutzle [21] even for t=30. It might be because of different machine environments used. When comparing the DDE_{RLS} algorithm to the IG_RS_{LS}, the DDE_{RLS} generated slightly better results for all t=30, t=60 and t=90. However, the success was due to the use of the referenced local search in the DDE algorithm.

We did not report the results for both algorithms without the local search versions. However, when no local search was employed, the performance of the IG_RS algorithm was superior to the DDE algorithm. One reason might be the fact that we have employed a very low mutation probability (0.2) indicating that the destruction and construction procedure is not so much effectively used in the DDE algorithm. When we increase the mutation probability to higher levels such as 0.8, the performance of the DDE algorithm becomes very competitive to IG_RS algorithm at the expense of limiting the performance of the DDE_{RLS} algorithm.

To sum up, the performance of the DDE_{RLS} algorithm was slightly better than our implementation of IG_RS_{LS} algorithm. However, the pure performance of the IG_RS algorithm was superior to the DDE algorithm. This could be compensated by increasing the mutation probability at the expense of reducing the impact of the RLS local search on the solution quality.

Table 1. IG_RS_{LS} Results for t=30

	ARPD				Time to Best Makespan			
Problem	Avg	Min	Max	Std	Avg	Min	Max	Std
20/5	0.03	0.00	0.04	0.02	0.04	0.00	0.11	0.04
20/10	0.01	0.00	0.03	0.01	0.35	0.09	0.73	0.27
20x20	0.02	0.00	0.05	0.02	1.21	0.20	2.62	0.97
50x5	0.00	0.00	0.01	0.00	0.25	0.03	0.59	0.24
50x10	0.48	0.37	0.63	0.11	2.33	0.40	4.81	1.81
50x20	0.75	0.51	0.95	0.18	8.97	4.56	12.56	3.31
100x5	0.01	0.00	0.01	0.00	0.77	0.09	1.78	0.67
100x10	0.23	0.16	0.29	0.05	4.05	1.30	7.51	2.53
100x20	1.04	0.76	1.27	0.23	17.26	8.74	26.46	7.19
200x10	0.15	0.06	0.25	0.10	9.61	1.66	19.68	8.08
200x20	1.13	0.92	1.34	0.17	33.09	12.66	52.07	16.68
500x20	0.67	0.55	0.81	0.10	90.93	36.92	135.02	42.38
Mean	0.38	0.28	0.47	0.08	14.07	5.55	21.99	7.01

Table 2. DDE_{RLS} Results for t=30

	ARPD				Time to Best Makespan			
Problem	Avg	Min	Max	Std	Avg	Min	Max	Std
20/5	0.04	0.04	0.04	0.00	0.06	0.02	0.17	0.06
20/10	0.02	0.00	0.04	0.02	0.39	0.08	0.81	0.31
20x20	0.03	0.00	0.08	0.04	1.17	0.11	2.45	1.01
50x5	0.00	0.00	0.01	0.01	0.54	0.06	1.15	0.47
50x10	0.49	0.29	0.67	0.17	2.72	1.02	5.02	1.76
50x20	0.74	0.45	1.01	0.22	9.27	4.39	12.94	3.48
100x5	0.00	0.00	0.00	0.00	0.34	0.08	0.80	0.30
100x10	0.15	0.06	0.22	0.08	4.82	1.12	9.36	3.59
100x20	1.11	0.81	1.42	0.26	18.56	10.50	27.27	6.90
200x10	0.06	0.05	0.09	0.02	9.54	3.82	18.83	6.37
200x20	0.99	0.70	1.19	0.21	39.87	21.88	57.11	15.57
500x20	0.50	0.41	0.57	0.07	105.89	54.73	143.34	36.84
Mean	0.35	0.23	0.45	0.09	16.10	8.15	23.27	6.39

Table 3. IG_RS_{LS} Results for t=60

	ARPD				Time to Best Makespan			
Problem	Avg	Min	Max	Std	Avg	Min	Max	Std
20/5	0.02	0.00	0.04	0.02	0.09	0.00	0.31	0.13
20/10	0.00	0.00	0.00	0.00	0.41	0.09	1.03	0.39
20x20	0.02	0.00	0.04	0.02	1.90	0.20	4.37	1.82
50x5	0.00	0.00	0.01	0.00	0.34	0.03	0.76	0.30
50x10	0.42	0.30	0.53	0.10	4.15	1.27	9.03	3.18
50x20	0.59	0.39	0.78	0.16	17.24	8.45	25.44	7.30
100x5	0.01	0.00	0.01	0.00	0.77	0.10	1.79	0.67
100x10	0.19	0.12	0.25	0.06	8.65	3.34	15.84	5.39
100x20	0.92	0.65	1.22	0.24	33.45	14.86	53.69	16.49
200x10	0.09	0.06	0.16	0.04	20.74	3.05	43.44	17.05
200x20	1.02	0.82	1.18	0.15	72.46	34.31	107.58	31.26
500x20	0.62	0.50	0.71	0.09	194.87	87.64	270.37	75.99
Mean	0.33	0.24	0.41	0.07	29.59	12.78	44.47	13.33

Table 4. DDE_{RLS} Results for t=60

	ARPD				Time to Best Makespan			
Problem	Avg	Min	Max	Std	Avg	Min	Max	Std
20/5	0.03	0.00	0.04	0.02	0.14	0.03	0.33	0.15
20/10	0.02	0.00	0.03	0.02	0.48	0.10	1.07	0.39
20x20	0.02	0.00	0.05	0.02	1.99	0.37	5.05	1.98
50x5	0.00	0.00	0.01	0.00	0.63	0.16	1.21	0.44
50x10	0.39	0.29	0.59	0.13	4.77	1.52	8.70	3.17
50x20	0.60	0.33	0.85	0.21	17.38	8.35	26.05	7.36
100x5	0.00	0.00	0.00	0.00	0.34	0.08	0.81	0.30
100x10	0.12	0.05	0.22	0.08	8.40	1.32	17.24	6.68
100x20	0.98	0.70	1.29	0.26	38.81	16.65	55.90	17.06
200x10	0.06	0.03	0.09	0.03	14.13	3.81	34.40	12.98
200x20	0.82	0.57	1.01	0.20	75.89	40.40	113.43	29.96
500x20	0.45	0.37	0.52	0.06	181.56	93.80	265.50	69.15
Mean	0.29	0.20	0.39	0.09	28.71	13.88	44.14	12.47

Table 5. IG_RS_{LS} Results for t=90

	ARPD				Time to Best Makespan			
Problem	Avg	Min	Max	Std	Avg	Min	Max	Std
20/5	0.02	0.00	0.04	0.02	0.09	0.00	0.31	0.13
20/10	0.00	0.00	0.00	0.00	0.41	0.09	1.03	0.39
20x20	0.02	0.00	0.04	0.02	2.09	0.20	5.17	2.14
50x5	0.00	0.00	0.01	0.00	0.33	0.03	0.76	0.30
50x10	0.41	0.30	0.53	0.10	5.39	1.27	11.53	4.28
50x20	0.54	0.32	0.72	0.16	1.90	9.69	34.11	9.81
100x5	0.00	0.00	0.01	0.00	1.12	0.09	3.50	1.43
100x10	0.17	0.08	0.24	0.07	12.24	3.50	26.70	9.56
100x20	0.84	0.62	1.03	0.18	49.14	22.12	78.15	23.53
200x10	0.07	0.05	0.10	0.02	29.28	3.99	60.02	23.72
200x20	0.93	0.72	1.12	0.16	118.72	42.58	171.79	54.92
500x20	0.58	0.44	0.69	0.10	273.61	100.92	405.69	127.78
Mean	0.30	0.21	0.38	0.07	42.86	15.37	66.56	21.50

Table 6. DDE_{RLS} Results for t=90

	ARPD				Time to Best Makespan			
Problem	Avg	Min	Max	Std	Avg	Min	Max	Std
20/5	0.03	0.00	0.04	0.02	0.14	0.03	0.33	0.15
20/10	0.01	0.00	0.03	0.02	0.61	0.10	1.46	0.57
20x20	0.02	0.00	0.04	0.02	2.92	0.38	7.82	3.18
50x5	0.00	0.00	0.01	0.00	0.81	0.16	1.84	0.67
50x10	0.34	0.29	0.40	0.05	8.10	2.07	14.75	5.28
50x20	0.55	0.32	0.76	0.18	22.99	11.31	35.54	10.02
100x5	0.00	0.00	0.00	0.00	0.34	0.08	0.80	0.30
100x10	0.08	0.05	0.14	0.05	13.86	3.86	30.76	11.33
100x20	0.83	0.54	1.06	0.22	57.98	24.42	82.78	24.19
200x10	0.05	0.03	0.06	0.01	20.11	3.92	56.16	21.43
200x20	0.77	0.55	0.98	0.20	106.84	52.20	168.26	49.22
500x20	0.42	0.35	0.50	0.07	262.95	104.43	407.39	127.63
Mean	0.26	0.18	0.34	0.07	41.47	16.91	67.32	21.16

During these runs (DDE_{RLS} runs), we were able to improve some best known solutions. New best known solutions for Taillard's benchmark instances are given below:

New best solution for *ta*051:
N=50, m=20, Cmax=3847
Permutation=

20	31	39	27	43	15	44
11	8	45	35	37	6	17
34	28	7	14	42	33	40
24	5	29	10	2	18	47
48	21	46	1	16	49	23
12	22	36	32	38	19	9
26	13	4	41	30	25	50
3						

New best solution for *ta*054:
N=50, m=20, Cmax=3719
Permutation=

5	21	11	14	36	30	13
24	12	7	45	19	35	20
31	25	37	3	44	33	32
50	48	43	49	29	46	23
10	40	15	38	9	17	42
22	6	39	26	47	4	27
18	8	2	41	34	1	16
28						

New best solution for *ta*056:
N=50, m=20, Cmax= 3680
Permutation=

14	37	3	18	8	33	11
21	42	5	13	49	50	20
28	45	43	41	46	15	24
44	40	36	39	4	16	47
17	27	1	26	10	19	32
25	30	7	2	31	6	23
48	22	29	34	9	35	38
12						

New best solution for *ta*059:
N=50, m=20, Cmax= 3741
Permutation=

3	14	8	37	22	32	12
46	16	9	41	30	38	24
10	1	18	17	34	50	28
36	40	29	26	47	6	7
13	27	33	39	23	11	49
45	4	5	43	48	21	31
42	19	25	2	20	15	44
35						

4. CONCLUSIONS

DE is a recent evolutionary optimization method. Besides the standard versions, we presented a new and novel discrete version denoted as DDE algorithm in this paper. Unlike the standard DE, the DDE algorithm is a novel algorithm employing a permutation representation for the problem on hand and works on a discrete domain. It indicates that it can be applied to all types of discrete combinatorial optimization problems in the literature. Furthermore, the DDE algorithm is hybridized with the referenced local search to further improve the solution quality.

The IG_RS and DDE algorithms were applied to the well-known benchmark problems of Taillard. The computational results show that the DDE_{LS} algorithm generated slightly better results than the IG_RS_{LS} algorithm. Ultimately, four instances are further improved for the well-known benchmarks of Taillard. However, the pure performance of the DPSO algorithm was not competitive to the pure IG_RS algorithm when no local search is employed in both algorithms.

As the future work, the authors have already solved the PFSP with a novel discrete particle swarm optimization algorithm. Extensive evaluation of DDE, DPSO and IG_RS with and without local search will be presented in the literature in the near future.

5. ACKNOWLEDGMENTS

We are grateful to Dr. Thomas Stützle for his generosity in providing the IG code. Even though we developed our own IG version in Visual C++, it was substantially helpful in grasping the IG algorithm in a great detail. We also appreciate his invaluable suggestions whenever needed.

6. REFERENCES

[1] Johnson, S. M. Optimal two-and three-stage production schedules. *Naval Research Logistics Quarterly, 1* (1954), 61-68.

[2] Rinnooy Kan, A. H. G. *Machine Scheduling Problems: Classification, Complexity, and Computations*. Nijhoff, The Hague, 1976.

[3] Palmer, D. S. Sequencing jobs through a multistage process in the minimum total time: A quick method of obtaining a near-optimum. *Operational Research Quarterly*, *16* (1965), 101-107.

[4] Campbell, H. G., Dudek, R. A., and Smith, M. L. A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, *16*, 10 (1970), B630-B637.

[5] Dannenbring, D. G. An evaluation of flow shop sequencing heuristics. *Management Science*, *23*, 11 (1977), 1174-1182.

[6] Nawaz, M., Enscore Jr., E. E., and Ham, I. A heuristic algorithm for the m-machine, n-job flow shop sequencing problem. *OMEGA*, *11*, 1 (1983), 91-95.

[7] Taillard, E. Some efficient heuristic methods for the flowshop sequencing problems. *European Journal of Operational Research*, *47* (1990), 65-74.

[8] Framinan, J. M., Leisten, R., and Ruiz-Usano, R. Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimization. *European Journal of Operational Research*, *141* (2002), 559-569.

[9] Framinan, J. M., and Leisten, R. An efficient constructive heuristic for flowtime minimisation in permutation flow shops. *OMEGA*, *31* (2003), 311-317.

[10] Rajendran, C., and Ziegler, H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, *155*, 2 (2004), 426-438.

[11] Stützle, T. An ant approach to the flowshop problem. In *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Cmputing (EUFIT'98)*, Verlag Mainz, Aachen, Germany, 1998, 1560-1564.

[12] Reeves, C. A genetic algorithm for flowshop sequencing. *Computers and Operations Research*, *22*, 1 (1995), 5-13.

[13] Reeves, C., and Yamada, T. Genetic algorithms, path relinking and the flowshop sequencing problem. *Evolutionary Computation*, *6* (1998), 45-60.

[14] Ruiz, R., Maroto, C., Alcaraz, J., 2006. Two new robust genetic algorithms for the flowshop scheduling problem. OMEGA, the International Journal of Management Science 34, 461–476.

[15] Stützle, T. *Applying iterated local search to the permutation flowshop problem*.Technical Report, AIDA-98-04, Darmstad University of Technology, Computer Science Department, Intellctics Group, Darmstad, Germany, 1998.

[16] Ogbu, F., and Smith, D. The application of the simulated annealing algorithm to the solution of the n/m/Cmax flowshop problem. *Computers and Operations Research*, *17*, 3 (1990), 243-253.

[17] Osman, I., and Potts, C. Simulated annealing for permutation flow shop scheduling. *OMEGA*, *17*, 6 (1989), 551-557.

[18] Nowicki, E., and Smutnicki, C. A fast tabu search algorithm for the permutation flowshop problem. *European Journal of Operational Research*, *91* (1996), 160-175.

[19] Watson, J. P., Barbulescu, L., Whitley, L. D., and Howe, A. E. Contrasting structured and random permutation flowshop scheduling problems: search space topology and algorithm performance. *ORSA Journal of Computing*, *14*, 2 (2002), 98-123.

[20] Grabowski, J., and Wodecki, M. A very fast tabu search algorithm for the permutation flowshop problem with makespan criterion. *Computers and Operations Research*, *31*, 11 (2004), 1891-1909.

[21] Ruiz, R., and Stützle, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, *177* (2007), 2033-2049.

[22] Ruiz, R., Maroto, C., 2005. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 165, 479–494.

[23] Taillard, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, *64* (1993), 278-285.

[24] Storn, R., and Price, K. Differential evolution - a simple and efficient heuristic for global optimization over continuous space. *Journal of Global Optimization*, *11* (1997), 341-359.

[25] Price, K., Storn, R., and Lampinen, J. *Differential Evolution – A Practical Approach to Global Optimization.* Springer-Verlag, 2006.

[26] Babu, B. V., and Onwubolu, G. C. (eds.) *New Optimization Techniques in Engineering.* Springer-Verlag, 2004.

[27] Al-Anzi, F. S., and Allahverdi, A. A self adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. *European Journal of Operational Research*, in press.

[28] Jacobs, L. W., and Brusco, M. J. A local search heuristic for large set-covering problems. *Naval Research Logistics Quarterly*, *42*, 7 (1995), 1129-1140.

[29] Marchiori, E., Steenbeek, A. An evolutionary algorithm for large set covering problems with applications to airline crew scheduling. In *Real-World Applications of Evolutionary Computing, EvoWorkshops 2000*, Lecture Notes in Computer Science, 1803, Springer-Verlag, Berlin, 2000, 367-381.

[30] Pan Q-K, Tasgetiren M. F, Liang Y-C, A Discrete Particle Swarm Optimization Algorithm for the No-Wait Flowshop Scheduling Problem with Makespan and Total Flowtime Criteria, Accepted to Bio-inspired metaheuristics for combinatorial optimization problems, Special issue of *Computers & Operations Research*, 2005.

[31] Rajendran, C. Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *International Journal of Production Economics*, *29* (1993), 65-73.

[32] Tasgetiren M. F., Yun-Chia Liang, Sevkli M., Gencyilmaz G, 2004, Differential evolution algorithm for permutation flowshop sequencing problem with makespan criterion *4th International Symposium on Intelligent Manufacturing Systems, IMS2004*, pp.442-452, September 5-8, 2004 ,Sakarya,Turkey

[33] Tasgetiren M. F., Yun-Chia Liang, Sevkli M., Gencyilmaz G, 2004, Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem., *International Journal of Production Research*, Vol. 44, No. 22/15, pp. 4737-4754, 2006

[34] Andreas C. Nearchou, Sotiris L. Omirou, Differential evolution for sequencing and scheduling optimization, *Journal of Heuristics*, Vol. 12, Issue 6, pp. 395-411, 2006

[35] Onwubolu Godfrey, Davendra Donald, Scheduling flow shops using differential evolution algorithm, *European Journal of Operational Research,* Vol. 171, No. 2, pp. 674-692