Enhanced Iterated Local Search Algorithms for the Permutation Flow Shop Problem Minimizing Total Flow Time

Xingye Dong[1,2], Maciek Nowak[2], Ping Chen[3] and Houkuan Huang[1]
[1]*School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China*
[2]*Quinlan School of Business, Loyola University, Chicago, IL 60611, U.S.A.*
[3]*TEDA College, NanKai University, Tianjin 300457, China*

Keywords: Scheduling, Permutation Flow Shop, Total Flow Time, Iterated Local Search, Neighborhood.

Abstract: Flow shop scheduling minimizing total flow time is a famous combinatorial optimization problem. Many algorithms have been proposed to solve it. Among them, iterated local search (ILS) is a simple, efficient and effective one. However, in existing ILS, one basic insertion neighborhood is generally used, greatly limiting the search space. In this work, an enhanced iterated local search (EILS) is proposed, using a hybrid of insertion and swap neighborhoods. The perturbation method also plays an important role in ILS. Two perturbation methods, the insertion method and a destruction and construction heuristic are tested in this paper. Both perform significantly better in comparison to three state of the art algorithms, indicating that the hybrid use of insertion and swap neighborhoods is effective for the discussed problem. However, there is no significant difference between the destruction and construction and the insertion perturbation methods.

1 INTRODUCTION

As a simple but efficient and effective metaheuristic, iterated local search (ILS) has attracted a great deal of attention (Lourenço et al., 2010). The framework of ILS is quite simple, with the pseudo code listed in Figure 1, where s^* denotes the best solution found so far. There are several key components in the ILS algorithm: the method generating the initial solution in step 1, the local search procedure in step 2, the acceptance criterion in step 3 and the method to perturb solution s in step 4.

```
1. Generate an initial solution $s$; let $s' = s$, $s^* = s$;
2. Generate $s'' = LocalSearch(s')$; update $s^*$ if better solution is found in the process;
3. Let $s = Accept(s, s'')$;
4. If the termination criterion is not satisfied, then generate $s' = Perturb(s)$ and go to step 2; otherwise, output $s^*$.
```

Figure 1: Pseudo code of an ILS framework.

The ILS algorithm has been applied to a variety of combinatorial optimization problems, including the permutation flow shop scheduling problem (PFSP). In the PFSP, there are n jobs requiring m operations to be performed on m machines in the same sequence, i.e., no pre-emption is allowed. All of the jobs are available for processing when the problem begins, the ith operation needs to be processed on the ith machine, each machine can serve at most one job at any time, and an operation can be processed only if the previous operation is completed and the requested machine is available. In literature, the solution of the PFSP is usually presented as a permutation of the jobs.

In practice, local search has been most frequently implemented as part of a more complex metaheuristic. The metaheuristics for solving the PFSP minimizing total flow time include the M-MMAS and PACO (Rajendran and Ziegler, 2004), the ACO1 and ACO2 (Rajendran and Ziegler, 2005), the differential evolution algorithm and iterated greedy algorithm (Pan et al., 2008), genetic algorithms (Tseng and Lin, 2009; Tseng and Lin, 2010; Zhang et al., 2009) and the artificial bee colony algorithm (Tasgetiren et al., 2011). Local search algorithms are also used independently and iteratively. For example, Dong et al. (Dong et al., 2009) propose an ILS algorithm to solve the PFSP minimizing total flow time, showing improvement over three constructive heuristics and four metaheuristics. More recently, Dong et al.

Dong X., Nowak M., Chen P. and Huang H..
Enhanced Iterated Local Search Algorithms for the Permutation Flow Shop Problem Minimizing Total Flow Time.
DOI: 10.5220/0004410100580065
In *Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2013), pages 58-65
ISBN: 978-989-8565-70-9

(Dong et al., 2013) presented the multi-restart ILS (MRSILS), improving the ILS by restarting the local search from a group of elite solutions. Their experiments show that the MRSILS is better than or comparable to five state of the art metaheuristics, as well as the ILS presented in (Dong et al., 2009). Finally, four independent local search methods are developed by Pan and Ruiz (Pan and Ruiz, 2012) and shown to be quite effective.

However, a defect in many current state of the art algorithms is a relatively limited search space. Quite often, only one neighborhood structure is used for local search, e.g., only an insertion neighborhood structure is used in the ILS algorithms by Dong et al. (Dong et al., 2009; Dong et al., 2013) and the algorithms by Pan and Ruiz (Pan and Ruiz, 2012). Using more than one neighborhood structure, as with variable neighborhood search (VNS), is an effective way to expand the search space. However, too many neighborhood structures can lead to a significant increase in computation time. Finding an appropriate balance between search space and computation time is vital.

In this work, an enhanced ILS (EILS) framework is proposed that uses both insertion and swap neighborhoods sequentially during the local search. The use of just one additional neighborhood structure opens up the search space considerably while limiting the increase in computational time. In order to further expand the search space, the EILS applies two perturbation methods, a simple insertion perturbation method and a destruction and construction heuristic. The insertion perturbation method is drawn from Dong et al. (Dong et al., 2013), and the destruction and construction heuristic is derived from an iterated greedy algorithm (Ruiz and Stützle, 2007), which has been successfully used by Pan and Ruiz (Pan and Ruiz, 2012).

The proposed EILS is evaluated on Taillard's set of benchmark problem instances (Taillard, 1993), and compared with the MRSILS developed by Dong et al. (Dong et al., 2013) and the ILS and IGA developed by Pan and Ruiz (Pan and Ruiz, 2012). These two algorithms have outperformed other algorithms on numerous data instances and they have established the current benchmark for comparison. While a great deal of research now focuses on more intricate and sophisticated metaheuristics, this work shows that a deeper analysis of the core problem through the utilization of a simple heuristic such as ILS has considerable potential to improve established methodologies.

The remainder of this paper is organized as follows. In Section 2, the formulation of the PFSP with total flow time criterion is presented. In Section 3, the EILS is described in detail. The computational results are illustrated and analyzed in Section 4, and the paper is concluded in Section 5.

2 PROBLEM FORMULATION

In this paper, the PFSP is discussed with the objective of minimizing total flow time. This problem is an important and well-known combinatorial optimization problem, first gaining attention through Johnson's pioneering work (Johnson, 1954). Many researchers focus their attention on this problem as it is considered more relevant to today's dynamic production environment (Liu and Reeves, 2001), as it tends to stabilize the use of resources and minimize the work-in-process inventory (Tseng and Lin, 2009).

In the PFSP, a set of jobs $J = \{1, 2, \ldots, n\}$ available at time zero must be processed on m machines, where $n \geq 1$ and $m \geq 1$. Each job has m operations, each of which has an uninterrupted processing time. The processing time of the ith operation of job j is denoted by p_{ij}, where $p_{ij} \geq 0$. The ith operation of a job is processed on the ith machine. An operation of a job is processed only if the previous operation of the job is completed and the requested machine is available. Each machine processes these jobs in the same order and at most one operation of each job can be processed at a time. This problem is usually denoted by $F_m|prmu|\sum C_j$ (Pinedo, 2001), where F_m describes the environment, $prmu$ is the set of constraints and C_j denotes the completion time of job j. Let π denote a permutation on the set J, representing a job processing order. Let $\pi(k)$, $k = 1, \ldots, n$, denote the kth job in π, then the completion time of job $\pi(k)$ on each machine i can be computed through a set of recursive equations:

$$C_{i,\pi(1)} = \textstyle\sum_{r=1}^{i} p_{r,\pi(1)} \qquad i = 1, \ldots, m \qquad (1)$$

$$C_{1,\pi(k)} = \textstyle\sum_{r=1}^{k} p_{1,\pi(r)} \qquad k = 1, \ldots, n \qquad (2)$$

$$C_{i,\pi(k)} = max\{C_{i-1,\pi(k)}, C_{i,\pi(k-1)}\} + p_{i,\pi(k)} \quad i = 2, \ldots, m; k = 2, \ldots, n \qquad (3)$$

Then $C_{\pi(k)} = C_{m,\pi(k)}$, $k = 1, \ldots, n$. The total flow time is $\sum C_{\pi(k)}$, or the sum of completion times on machine m for all jobs. The objective of the PFSP when minimizing total flow time is to minimize $\sum C_{\pi(k)}$, or C_π for short. This problem is NP-complete even with only two machines (Garey et al., 1976).

3 ENHANCED ITERATED LOCAL SEARCH

As aforementioned, there are four key components in an ILS algorithm. In this section, three key components, initial method, local search procedure and perturbation method, are discussed and the framework of the EILS is illustrated.

3.1 Initialization Method

In literature, a good heuristic is often used as the initialization method. The H(x) heuristics by Liu and Reeves (Liu and Reeves, 2001), which rank jobs according to the ascending order of a defined index function reflecting weighted total machine idle time and artificial flowtime, are highly effective for the this purpose. H(x) heuristics are used to generate initial solutions for several state of the art metaheuristics, such as the four local-search-based algorithms by Pan and Ruiz (Pan and Ruiz, 2012). Dong et al. (Dong et al., 2009) used the H(2) method to generate an initial solution in negligible time, allowing for the ILS algorithm to perform very well. H(2) is also used in the MRSILS algorithm (Dong et al., 2013). In this work, the H(2) heuristic is used to generate initial solutions.

3.2 Local Search Procedure

An efficient local search procedure is very important to the success of ILS as it mainly determines the solution space that may be explored. Driving the search to a promising area is vital to an ILS algorithm, such as with the MRSILS, which is successful because it extends the search space by generating new start solutions from a set of elite solutions. However, the search space can also be extended through the use of a local search based on a variety of neighborhood structures.

In this work, two commonly used local search methods are jointly applied, insertion and swap. The neighborhood of an insertion local search is all solutions that may be found by moving a job and inserting it elsewhere. This is described in Figure 2, where π^* denotes the best solution found so far, $flag = true$ indicates that π^* is improved during the search, idx denotes the index of the inserting job in π_{seq}, and each of these is a global variable. In this local search, each of the n jobs is inserted into each of the other $n-1$ positions and the current solution is updated when a better neighbor is found. The job that is being inserted is denoted by $\pi_{seq}(idx+1)$. When the search is trapped in a local optimum, such that $cnt = n$ in line 3, it is stopped and the current solution π is returned. The search sequence is determined by π_{seq}, which equals the start solution π at the beginning, and then updated when the π^* is improved (line 11). This search sequence scheme is derived from the idea of the referenced index search method (Rajendran and Ziegler, 1997).

```
    Insertion_LS($\pi$)
1.  {
2.      Set $\pi_{seq} = \pi$, $cnt = 0$;
3.      while($cnt < n$)
4.      {
5.          Find $j$, where $\pi(j) = \pi_{seq}(idx+1)$;
6.          Insert $\pi(j)$ to other $n-1$ positions,
            respectively, and let $\pi'$ be the best
            generated solution;
7.          if($C_{\pi'} < C_\pi$)
8.          {
9.              $cnt = 1, \pi = \pi'$;
10.             if($C_\pi < C_{\pi^*}$)
11.                 $\pi^* = \pi$, $flag = true$, $\pi_{seq} = \pi$;
12.         }
13.         else
14.             $cnt++$;
15.         $idx = (idx+1) \bmod n$;
16.     }
17.     return $\pi$;
18. }
```

Figure 2: Pseudo code of the insertion local search.

The neighborhood of a swap local search is those solutions that may be found by swapping the position of two jobs in the sequence. This is illustrated in Figure 3, where the π^*, $flag$ and idx have the same meaning as in the insertion local search. The job being swapped is denoted by $\pi_{seq}(idx+1)$. This neighborhood search swaps each job with earlier jobs in the current solution, and the current solution is updated when a better neighbor is found. When the search is trapped in a local optimum, such that $cnt = n$ in line 4, it is stopped and the current solution π is returned. The search sequence is also determined by π_{seq}, which equals the start solution π throughout the procedure. In this search, idx always starts from zero.

There are $(n-1)^2$ neighbors in an insertion neighborhood, while $n \times (n-1)/2$ neighbors in a swap neighborhood, given that swapping job a with b is the same as swapping b with a. Since the computational complexity to evaluate one solution is $O(nm)$, the computational complexity for evaluating all the neighbors with both local search methods is at least $O(n^3m)$.

```
    Swap_LS($\pi$)
 1. {
 2.     Set $\pi_{seq} = \pi$, $cnt = 0$;
 3.     Set global variable $idx = 0$;
 4.     while($cnt < n$)
 5.     {
 6.         Find $j$, where $\pi(j) = \pi_{seq}(idx+1)$;
 7.         Swap $\pi(j)$ with the jobs earlier
            in the sequence, and let $\pi'$ be
            the best generated solution;
 8.         if($C_{\pi'} < C_\pi$)
 9.             $cnt = 1$, $\pi = \pi'$;
10.         else
11.             $cnt++$;
12.         $idx = (idx+1) \bmod n$;
13.     }
14.     if($C_\pi < C_{\pi^*}$)
15.         $\pi^* = \pi$, $flag = true$;
16.     return $\pi$;
17. }
```

Figure 3: Pseudo code of the swap local search.

3.3 Perturbation Method

When the local search procedure is trapped in a local optimum, a new start solution is generated in order to continue the local search. The new start solution is often generated by perturbing an elite solution, such that the perturbation method plays an important role in the success of an ILS algorithm. Insertion and swap perturbation are two commonly used methods. In literature, several researchers have reported that the former performs better than the latter (Tasgetiren et al., 2011; Dong et al., 2013) and this paper focuses on insertion perturbation. The insertion perturbation is described in Figure 4, where *idx* is a global variable, denoting the start index of the job to be inserted in the next iteration of local search. After perturbation, *idx* is set according to the insertion position.

The variable *idx* is used to avoid cycling, particularly when the new start solution is generated through insertion perturbation. For example, suppose an elite solution $\pi^* = (1,2,3,4,5)$ is selected for perturbation to create a new start solution $\pi = (4,1,2,3,5)$. If job 4 is the first repositioned job, then it will be reinserted to position 4 as π^* is a local optimum. In order to avoid cycling, the job that was moved to generate the new start solution should be the last one selected in the following local search.

Another perturbation method comes from the iterated greedy (IG) algorithm, which is introduced by Ruiz and Stützle (Ruiz and Stützle, 2007) to solve

```
    Insertion_Perturb($\pi$)
 1. {
 2.     Randomly select a job in $\pi$ and remove
        it from the sequence; Insert the job into
        any randomly chosen position (except
        in its original position $j$) ;
 3.     Set $idx = j \bmod n$, $\pi_{seq} = \pi$;
 4.     return $\pi$;
 5. }
```

Figure 4: Pseudo code of the insertion perturbation method.

the PFSP minimizing makespan. The IG algorithm runs iteratively over a destruction and construction procedure. In the destruction stage, several jobs are removed randomly from a solution to form a partial solution, after which the jobs are reinserted sequentially in a greedy way to reconstruct a full solution. This method has been used by Pan and Ruiz (Pan and Ruiz, 2012) as a perturbation method in their IGA. In this work, the destruction and construction procedure is also used as a perturbation method. It is illustrated in Figure 5, where d is the number of removed jobs, reflecting the perturbation strength, while π^R is a pool for the removed jobs, and the first job in this pool is denoted by $\pi^R(1)$. In the work of Pan and Ruiz (Pan and Ruiz, 2012), d is set to 8, which is the value used here. After perturbation, the global variable *idx* is set to zero.

```
    Destruct_Construct($\pi$, $d$)
 1. {
 2.     Define $\pi^R$, $i = 0$;
 3.     while($i < d$)
 4.     {
 5.         Delete a job from $\pi$ randomly, and
            add the job to the end of $\pi^R$;
 6.         Set $i = i+1$;
 7.     }
 8.     while($i > 0$)
 9.     {
10.         Insert $\pi^R(1)$ back to $\pi$ to the posi-
            tion where the new partial solution
            $\pi$ is optimized;
11.         Delete $\pi^R(1)$ from $\pi^R$, set $i = i-1$;
12.     }
13.     Set $idx = 0$;
14.     return $\pi$;
15. }
```

Figure 5: Pseudo code of the destruction and construction perturbation method.

3.4 Framework of the proposed Enhanced ILS

The framework of the full EILS is illustrated in Figure 6, where π^* denotes the best solution during the search, *pool* indicates a set of elite solutions which is empty at the start.

```
1. Define global variable $\pi^*$, $flag$ and $idx$;
2. Generate an initial solution $\pi$; Set $pool = \emptyset$,
   $pool\_size$; Let $\pi^* = \pi$, $idx = 0$;
3. while(termination criterion is not satisfied)
4. {
5.     $flag = false$;
6.     $\pi$ = Insertion_LS($\pi$);
7.     $\pi$ = Swap_LS($\pi$);
8.     if($flag$)
9.         Set $pool = \emptyset$;
10.    if($\pi \notin pool$)
11.        Add $\pi$ to $pool$;
12.    if($|pool| > pool\_size$)
13.        Delete the worst solution in $pool$;
14.    if($|pool| < pool\_size$)
15.        $\pi = \pi^*$;
16.    else
17.        Randomly choose one solution from
18.        $pool$ as $\pi$;
19.    Perturb $\pi$ to generate a new $\pi$;
20.}
21.Output $\pi^*$ and stop.
```

Figure 6: Pseudo code of the proposed EILS.

In literature, there are usually two termination criteria: a maximum number of iterations and limited computational time. In this work, the latter is applied on line 3 in order to more easily compare the EILS with other algorithms. The insertion local search and swap local search are used sequentially in line 6 and line 7. If the best solution π^* is improved during the local search, the elite solution set *pool* is emptied; otherwise, the local optimum π is added to *pool* as an elite solution if it is not in *pool* yet, and the worst solution in *pool* is erased if the size of *pool* is larger than its limitation *pool_size*. In this work, the parameter *pool_size* is set to 5 as the value is found most effective in Dong et al. (Dong et al., 2013), though it is quite robust. This pooling strategy is drawn from Dong et al. (Dong et al., 2013).

If *pool* is not full, the best solution found so far, π^*, is chosen to generate a new start solution π by the specified perturbation method. While *pool* is full, a solution is randomly chosen from *pool* and perturbed by the specified perturbation method to generate a new start solution π. A version of EILS is implemented for both insertion perturbation and destruction and construction perturbation, denoted by EILS-INS and EILS-DC, respectively. If the insertion perturbation is used and the swap local search (line 7) is removed, EILS is the same as MRSILS (Dong et al., 2013), except for the adjustment in the search sequence. In order to inspect the effect of the adjusted search sequence, this version of EILS is denoted by EILS-NoSwap and compared with MRSILS in the next section.

EILS is similar to a variable neighborhood search algorithm; however, in a variable neighborhood search algorithm, the search is restarted from the first neighborhood when the best solution found so far is improved (Lourenço et al., 2010). Regardless of whether the best solution found so far is improved with local search in EILS, the algorithm will continue into the perturbation stage.

4 COMPUTATIONAL RESULTS

In this section, EILS is compared with three state of the art metaheuristics, MRSILS by Dong et al. (Dong et al., 2013) and two of the four algorithms by Pan and Ruiz (Pan and Ruiz, 2012). These are some of the best available metaheuristics for this version of the PFSP, producing the highest quality solutions available.

MRSILS by Dong et al. (Dong et al., 2013) performs better than the ILS algorithm developed by Dong et al. (Dong et al., 2009), the discrete differential evolution algorithm and the iterated greedy algorithm by Pan et al. (Pan et al., 2008), the hybrid genetic algorithm by Zhang et al. (Zhang et al., 2009), and the discrete artificial bee colony and discrete differential evolution algorithms by Tasgetiren et al. (Tasgetiren et al., 2011).

Pan and Ruiz present four local search based algorithms, one based on the idea of iterated local search (denoted as PR-ILS) and another on iterated greedy algorithm (denoted as PR-IGA), with each extended to a population-based algorithm. PR-ILS and PR-IGA perform better than the population-based versions, as well as 12 other powerful metaheuristics published in recent years. In this work, PR-ILS and PR-IGA are re-implemented with the same parameters as used by Pan and Ruiz and compared with EILS. The local search procedure used in PR-ILS and PR-IGA is insertion-based, with a computational complexity of at least $O(n^3m)$.

Each algorithm is implemented in C++, running on a PC with an Intel Core2 Duo processor (2.99 GHz) with 2G main memory. Though the com-

Table 1: Comparison in ARPD for MRSILS, PR-ILS, PR-IGA, EILS_NoSwap, EILS_INS and EILS_DC ($0.004 \times n^3 \times m$ ms CPU time).

$n\|m$	MRSILS	PR-ILS	PR-IGA	EILS_NoSwap	EILS_INS	EILS_DC
20|5	0.007	0.007	0.007	0	0	0
20|10	0	0.002	0	0.004	0.006	0
20|20	0	0	0	0	0	0
50|5	0.339	0.411	0.373	0.342	0.298	0.303
50|10	0.495	0.467	0.436	0.337	0.407	0.391
50|20	0.284	0.378	0.375	0.314	0.405	0.241
100|5	0.328	0.410	0.340	0.350	0.218	0.242
100|10	0.464	0.443	0.513	0.473	0.346	0.523
100|20	0.484	0.540	0.560	0.558	0.387	0.515
Avg.	0.267	0.295	0.289	0.264	0.230	0.246

Table 2: Comparison in ARPD for MRSILS, PR-ILS, PR-IGA, EILS_NoSwap, EILS_INS and EILS_DC ($0.012 \times n^3 \times m$ ms CPU time).

$n\|m$	MRSILS	PR-ILS	PR-IGA	EILS_NoSwap	EILS_INS	EILS_DC
20|5	0.007	0	0	0.007	0	0
20|10	0	0	0	0	0.002	0
20|20	0	0	0	0	0	0
50|5	0.203	0.255	0.233	0.211	0.128	0.139
50|10	0.240	0.354	0.383	0.329	0.259	0.212
50|20	0.209	0.238	0.281	0.176	0.251	0.168
100|5	0.179	0.291	0.191	0.131	0.031	0.073
100|10	0.234	0.308	0.330	0.183	0.123	0.203
100|20	0.289	0.161	0.259	0.153	0.239	0.188
Avg.	0.151	0.179	0.186	0.132	0.115	0.109

puter has two processors, only one is used in the experiments, as no parallel programming technique has been applied. In order to show the performance with increased CPU time, the CPU time is limited to $t \times n^3 \times m$ milliseconds, where t equals 0.004, 0.012 and 0.02, respectively.

The benchmark instances used for analysis are taken from Taillard (Taillard, 1993), with 120 instances evenly distributed among 12 different sizes. The scale of these problems varies from 20 jobs and 5 machines to 500 jobs and 20 machines. In the literature, most metaheuristics are tested on the first 90 benchmark instances (Pan et al., 2008; Tseng and Lin, 2009; Tseng and Lin, 2010; Zhang et al., 2009; Tasgetiren et al., 2011; Dong et al., 2009; Dong et al., 2013), as the largest instances with 200 and 500 jobs are too time consuming. This study also uses the first 90 instances. For each algorithm, five independent runs are performed on each benchmark instance and the best solution among the five runs is recorded. The relative percentage deviation (RPD) is calculated as:

$$RPD = (F - F_{best})/F_{best} \times 100 \qquad (4)$$

where F is the result found by the algorithm being evaluated and F_{best} is the best result provided by Dong et al. (Dong et al., 2013), and it is obtained by running the MRSILS ten independent times with an extended time period of $0.4 \times n \times m$ seconds for each instance and the best among these ten runs is chosen as the result.

The average RPD (ARPD) for each combination of n and m and the ARPD for all 90 instances are presented in Table 1 - 3. These tables show that on average EILS-INS and EILS-DC perform the best, with EILS-NoSwap, MRSILS, PR-ILS and PR-IGA following in that order.

Compared with MRSILS, EILS-NoSwap only differs in that it has an adjusted search sequence, which can avoid cycling. From the experimental results, EILS-NoSwap performs slightly better than MRSILS on average, showing the effectiveness of the adjustment. However, the difference is not significant. EILS-INS builds on EILS-NoSwap with the addition of the swap local search. On average, the ARPD is lower for EILS-INS, indicating that this local search has considerable benefit. While several researchers have reported that the swap local search is not better than the insertion local search (Tasgetiren et al., 2011; Dong et al., 2013), this experiment shows that it is more effective when the two are combined to expand the search space.

EILS-INS performs slightly better than EILS-DC

Table 3: Comparison in ARPD for MRSILS, PR-ILS, PR-IGA, EILS_NoSwap, EILS_INS and EILS_DC ($0.02 \times n^3 \times m$ ms CPU time).

$n\|m$	MRSILS	PR-ILS	PR-IGA	EILS_NoSwap	EILS_INS	EILS_DC
20|5	0.007	0	0.007	0	0	0
20|10	0	0	0	0	0	0
20|20	0	0	0	0	0	0
50|5	0.155	0.191	0.228	0.159	0.160	0.074
50|10	0.192	0.209	0.290	0.167	0.226	0.178
50|20	0.130	0.240	0.151	0.137	0.189	0.062
100|5	0.128	0.203	0.156	0.067	-0.075	0.039
100|10	0.200	0.325	0.238	0.137	0.042	0.147
100|20	0.007	0.096	0.258	0.106	0.004	0.122
Avg.	0.091	0.141	0.148	0.086	0.061	0.069

with the shortest and longest computational times, while EILS-DC is slightly better with a CPU time of $0.012 \times n^3 \times m$ ms. This indicates that the more complex destruction and construction perturbation method is not necessarily better than the simpler insertion perturbation method. The average RPDs for PR-ILS and PR-IGA confirm this, with PR-IGA performing slightly better than PR-ILS when computational time is more limited, while it performs slightly worse when the time is extended. For the most part, this is in accordance with Pan and Ruiz (Pan and Ruiz, 2012), where the authors show that PR-IGA performs slightly better than PR-ILS, but not significantly.

5 CONCLUSIONS

In this paper, an enhanced iterated local search framework is proposed to solve the permutation flow shop scheduling problem minimizing total flow time, an intensively studied combinatorial optimization problem. In EILS, two commonly used local search methods, insertion local search and swap local search, are used. Further, two different perturbation methods are applied. EILS searches the solution space intensively near a group of elite solutions. Compared with some sophisticated metaheuristics, EILS is simple and easy to implement.

Comparisons on Taillard's benchmarks show that both EILS versions, EILS-INS and EILS-DC, are significantly better than three state-of-the-art algorithms. The improved performance is mainly due to the combined use of the two neighborhood searches. In the literature, several researchers have reported that the swap local search is not better than the insertion local search, and so only the insertion local search is used. This work shows that the joint use of insertion and swap local search results in a more powerful local search. Applying this combined strategy to other metaheuristics may be beneficial, while it may also be helpful with other combinatorial problems. This work indicates that the local search-based algorithm deserves deeper study.

ACKNOWLEDGEMENTS

This work is supported by The Fundamental Research Funds for the Central Universities of China (Project Ref. 2009JBM018, Beijing Jiaotong University).

REFERENCES

Dong, X., Chen, P., Huang, H., and Nowak, M. (2013). A multi-restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time. *Computers and Operations Research*, 40:627–632.

Dong, X., Huang, H., and Chen, P. (2009). An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Computers and Operations Research*, 36:1664–1669.

Garey, M., Johnson, D., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1:117–129.

Johnson, S. (1954). Optimal two and three-stage production schedule with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68.

Liu, J. and Reeves, C. (2001). Constructive and composite heuristic solutions to the $p//\sum c_i$ scheduling problem. *European Journal of Operational Research*, 132:439–452.

Lourenço, H., Martin, O., and Stützle, T. (2010). *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, chapter Iterated Local Search: Framework and Applications, pages 363–397. Springer US.

Pan, Q.-K. and Ruiz, R. (2012). Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, 222:31–43.

Pan, Q.-K., Tasgetiren, M., and Liang, Y.-C. (2008). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers and Industrial Engineering*, 55:795–816.

Pinedo, M. (2001). *Scheduling: theory, algorithms, and systems*. Prentice Hall, 2nd edition.

Rajendran, C. and Ziegler, H. (1997). An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. *European Journal of Operational Research*, 103:129–138.

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

Rajendran, C. and Ziegler, H. (2005). Two ant-colony algorithms for minimizing total flowtime in permutation flowshops. *Computers and Industrial Engineering*, 48:789–797.

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

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285.

Tasgetiren, M., Pan, Q.-K., Suganthan, P., and Chen, A. H.-L. (2011). A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. *Information Sciences*, 181:3459–3475.

Tseng, L.-Y. and Lin, Y.-T. (2009). A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 198:84–92.

Tseng, L.-Y. and Lin, Y.-T. (2010). A genetic local search algorithm for minimizing total flowtime in the permutation flowshop scheduling problem. *International Journal of Production Economics*, 127:121–128.

Zhang, Y., Li, X., and Wang, Q. (2009). Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. *European Journal of Operational Research*, 196(3):869–876.