A Self-adaptive Iterated Local Search Algorithm on the Permutation Flow Shop Scheduling Problem

Xingye Dong[1], Maciek Nowak[2], Ping Chen[3] and Youfang Lin[1]
[1]*Beijing Key Lab of Traffic Data Analysis and Mining, School of Computer and IT, 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, Self-adaptive Perturbation.

Abstract: Iterated local search (ILS) is a simple, effective and efficient metaheuristic, displaying strong performance on the permutation flow shop scheduling problem minimizing total flow time. Its perturbation method plays an important role in practice. However, in ILS, current methodology does not use an evaluation of the search status to adjust the perturbation strength. In this work, a method is proposed that evaluates the neighborhoods around the local optimum and adjusts the perturbation strength according to this evaluation using a technique derived from simulated-annealing. Basically, if the neighboring solutions are considerably worse than the best solution found so far, indicating that it is hard to escape from the local optimum, then the perturbation strength is likely to increase. A self-adaptive ILS named SAILS is proposed by incorporating this perturbation strategy. Experimental results on benchmark instances show that the proposed perturbation strategy is effective and SAILS performs better than three state of the art algorithms.

1 INTRODUCTION

Since the pioneering work of Johnson (Johnson, 1954), the permutation flow shop problem (PFSP) has attracted considerable attention. In this problem, there are n jobs and m machines, and each job has m operations. The jobs need to be processed on m machines in the same sequence, that is to say no preemption is allowed. The ith operation of each job needs to be processed on the ith machine. All the jobs are available at time zero and each machine can serve at most one job at any time. Any operation can be processed only if its previous operation has been processed and the requested machine is available. The PFSP is NP-complete when minimizing total flow time with more than one machine (Garey et al., 1976).

For the purpose of finding high-quality solutions within a reasonable computation time, many methods, including simple heuristics and more complex metaheuristics (Dong et al., 2013), have been proposed. Among the existing metaheuristics, local search procedure plays an important role. Several ant colony algorithms are proposed by Rajendran et al. (Rajendran and Ziegler, 2004; Rajendran and Ziegler, 2005), working with well designed local search procedures and performing better than some constructive heuristics by Liu and Reeves (Liu and Reeves, 2001). Tasgetiren et al. (Tasgetiren et al., 2007) apply a particle swarm optimization (PSO) algorithm by using the smallest position value rule. They also propose a hybrid algorithm with variable neighborhood search (VNS), called PSO_{VNS}, and it performs quite well on Taillard's benchmark instances (Taillard, 1993). Pan et al. (Pan et al., 2008) develop two metaheuristics, a differential evolution algorithm and an iterated greedy algorithm hybridized with a referenced local search procedure. Local search procedures are also used in several genetic algorithms (Tseng and Lin, 2009; Zhang et al., 2009; Tseng and Lin, 2010). Recently, Tasgetiren et al. (Tasgetiren et al., 2011) designed an artificial bee colony algorithm and a discrete differential evolution algorithm. Both algorithms use a local search procedure taking advantage of the iterated local search (ILS) by Dong et al. (Dong et al., 2009) and the iterated greedy (IG) algorithm by Ruiz and Stützle (Ruiz and Stützle, 2007). An asynchronous genetic local search algorithm, embedding an enhanced variable neighborhood search, is also addressed by Xu et al. (Xu et al., 2011) for the PFSP minimizing total

Dong X., Nowak M., Chen P. and Lin Y..
A Self-adaptive Iterated Local Search Algorithm on the Permutation Flow Shop Scheduling Problem.
DOI: 10.5220/0005092003780384
In *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2014), pages 378-384
ISBN: 978-989-758-039-0

flow time.

Local search procedure is used as an embedded procedure in the aforementioned metaheuristics. However, it is also used in an iterative way to form a metaheuristic called iterated local search (ILS). Dong et al. (Dong et al., 2009) develop an ILS to solve the PFSP minimizing total flow time, in which the perturbation method swaps several pairs of adjacent jobs and the perturbation strength, denoted by the number of swapping pairs, is evaluated. A multi-restart iterated local search (MRSILS) algorithm is proposed by Dong et al. (Dong et al., 2013), improving the perturbation method mainly by generating restart solutions from a set of elite solutions. Their experiments show that the MRSILS increases the performance of the methodology used in Dong et al. (Dong et al., 2009) significantly, while performing comparably to or better than five other state of the art metaheuristics (Pan et al., 2008; Zhang et al., 2009; Tasgetiren et al., 2011). Costa et al. (Costa et al., 2012b) study the combination of the most commonly used local search neighborhoods, the swap neighborhood and the insertion neighborhood. Six different combinations in total are calibrated. Later, they extend this work by developing an algorithm hybridizing VNS and path-relinking on a particle swarm framework, with promising experimental results on Taillard's benchmark instances (Costa et al., 2012a). Pan and Ruiz (Pan and Ruiz, 2012) propose two local search methods based on the well known ILS and IG frameworks. They also extend them to population-based versions; however, their experiments show that the two local search methods perform better than the population-based versions.

Though ILS has performed well on the PFSP minimizing total flow time, one limitation is that the search process often cannot improve the best solution, even with dozens of iterations. For example, this occurs with the MRSILS by Dong et al. (Dong et al., 2013), although the pooling strategy in this algorithm augments the ability to find better solutions. A reason for the search process to be limited is that the perturbation method moves randomly selected jobs to other randomly selected positions without considering the search status. In order to improve perturbation quality, this work proposes a method to evaluate the convergence status of the search. With a greater focus on convergence, the probability increases that a job is moved to a position where a solution with larger objective can be generated. Based on this observation, a self-adaptive ILS (SAILS) is proposed and evaluated. Comparison results with the MRSILS by Dong et al. (Dong et al., 2013) and two local search based algorithms by Pan and Ruiz (Pan and Ruiz, 2012) on Taillard's benchmark instances (Taillard, 1993) show that the new perturbation method leads to improved performance.

The remainder of this paper is organized as follows. In Section 2, the formulation of the PFSP with total flow time criterion is presented. Section 3 describes the evaluation method and illustrates the proposed algorithm. The evaluation method is analyzed and SAILS is compared with several state of the art algorithms in Section 4, then 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. 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. The PFSP to minimize total flow time 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:

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

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

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

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.

3 THE PROPOSED ALGORITHM

According to Lourenco et al. (Lourenço et al., 2010), the general framework of ILS has four key compo-

nents: the method generating the initial solution, the local search procedure, the acceptance criterion and the method to perturb a solution. In this work, the H(2) heuristic by Liu and Reeves (Liu and Reeves, 2001) is used to generate the initial solution, as it can generate an initial solution in negligible time and has been used by Dong et al. (Dong et al., 2013; Dong et al., 2009) to form quite good ILS algorithms. As for the other three key components, the local search procedure and the evaluation method used to indicate the convergence status are discussed in Section 3.1. The acceptance criterion and the perturbation method are addressed together in Section 3.2. Finally, the proposed ILS algorithm is illustrated in Section 3.3.

3.1 Evaluation Method and Local Search

In this work, a solution to the discussed problem is presented as a permutation of the jobs and the commonly used insertion local search is chosen as the local search procedure. During the local search, each job is removed from its original position and inserted into the other $n-1$ positions to see whether the solution can be improved. Suppose the current local optimum solution is π and $f_{\pi(i),j}$ denotes the objective value of the solution generated by removing job $\pi(i)$ and inserting it into position j. Among the $n-1$ solutions, the best one is chosen and its objective is denoted by $f^*_{\pi(i)}$. The objective values form a matrix F, called the objective matrix. For all of the jobs, the average objective value of $f^*_{\pi(i)}, i = 1,\ldots,n$ can be computed as:

$$avg_\pi = \frac{1}{n}\sum_{i=1}^{n} f^*_{\pi(i)}. \quad (4)$$

This average objective value can be used as an indicator of the convergence status. Suppose the current best objective value in the search process is f_b. The larger the difference $avg_\pi - f_b$, the worse the solutions in the neighborhood surrounding π and the more difficult it is to escape from the local minimum in this neighborhood. This is an indicator that the perturbation strength should be increased. In order to temper the influence of this indicator as the difference between avg_π and f_b becomes larger, this difference is adjusted such that:

$$avg'_\pi = (avg_\pi - f_b)^{1/k}. \quad (5)$$

where k is an integer, and its value is tuned in Section 4.

The local search used in this work is shown in Figure 1, where π denotes the start solution, π^* denotes the best solution found so far in the search process, and F_π and F_{π^*} denote the objective matrices corresponding to π and π^*, respectively.

```
procedure Insertion_LS(π)
1.   cnt ← 0, idx ← 0, π_seq ← π*;
2.   while (cnt < n) do
3.     Find j, where π(j) = π_seq(idx+1);
4.     Move π(j) to other n−1 positions in
       π, respectively; denote the best solut-
       ion as π'; update F_π concurrently;
5.     if C_π' < C_π then
6.       π ← π', cnt ← 0;
       else
7.       cnt ← cnt + 1;
       endif
8.     if C_π < C_π_seq then
9.       π_seq ← π;
       endif
10.    idx ← (idx+1) mod n;
     endwhile
11.  return π;
   end
```

Figure 1: Pseudo code of the insertion local search.

3.2 Acceptance and Perturbation Method

In the general framework of the ILS (Lourenço et al., 2010), a decision is made whether to accept a solution as the local optimum when it is reached, and then the solution may be perturbed to generate a restart solution that continues the local search procedure. Dong et al. (Dong et al., 2013) propose a pooling strategy that leads the search to a more promising solution space, generating highly competitive solutions. In this work, the pooling strategy is also used, while the perturbation method is adapted by using the indicator avg'_π and incorporating the concept of temperature control from simulated annealing (Nikolaev and Jacobson, 2010).

According to Dong et al. (Dong et al., 2013), a set of elite solutions is pooled during the local search. Suppose the set of elite solutions is *pool* and π is the solution chosen from it. It is perturbed by randomly selecting a job, i, and moving it to another randomly selected position with probability $exp(-avg'_\pi/T)$, where T denotes temperature. In this work, T is a constant value. Alternatively, with probability $1-exp(-avg'_\pi/T)$ the randomly selected job is moved to a position that can generate a worse solution. The probability of moving job i to position j, where $j \neq i$, is denoted by p_j and can be determined

procedure Perturbation()
1. **if** $|pool| < pool\_size$ **then**
2. $\pi \leftarrow \pi^*$, $F_\pi \leftarrow F_{\pi^*}$;
   else
3. Choose a solution from *pool* as π, get the corresponding F_π from F_{pool};
   endif
4. Compute avg'_π using F_π;
5. Randomly select a job $\pi(i)$ from π;
6. **if** $rand() < exp(-avg'_\pi/T)$ **then**
7. Move $\pi(i)$ to another randomly selected position in π to form a new π;
   else
8. Move $\pi(i)$ to another position in π according to Eq. (6) to form a new π;
   endif
9. **return** π;

end

Figure 2: Pseudo code of the perturbation method.

Table 1: Global variables used in the perturbation method.

variables	Description
π^*	The best solution found so far
π	A variable denotes a solution
pool	The set of elite solutions
pool_size	The size limitation of *pool*
F_π	Objective matrix corresponding to π
F_{π^*}	Objective matrix corresponding to π^*
F_{pool}	Pool of objective matrix corresponding to *pool*

using a roulette methodology, such that the probability increases with the cost of the solution generated by the move. This probability is determined as:

$$p_j = \frac{\sqrt{f_{\pi(i),j} - f_b + 1}}{\sum_{k=0;k\neq i}^{n} \sqrt{f_{\pi(i),k} - f_b + 1}} \quad (6)$$

The perturbation method is illustrated in Figure 2. In this figure, $rand()$ is a function that generates a random number uniformly distributed in the range [0, 1], and the global variables are listed in Table 1.

3.3 Self-adaptive ILS

The proposed ILS is named self-adaptive ILS (SAILS), with the pseudo code presented in Figure 3.

In this work, the *pool_size* is set to 5, as this value is found to be effective in Dong et al. (Dong et al., 2013), although the parameter is rather robust. The parameter k (used in Eq. (5)) and temperature T are tuned in Section 4.1. There are usually two termination criteria in the literature: a maximum number of iterations and limited computational time. In this

1. Define *pool_size*, k and T;
2. $\pi \leftarrow$ Generate an initial solution;
3. Initialize F_π by setting every entry to C_π;
4. $\pi^* \leftarrow \pi$; $F_{\pi^*} \leftarrow F_\pi$; $pool \leftarrow \emptyset$; $F_{pool} \leftarrow \emptyset$;
5. **while**(termination criterion is not satisfied) **do**
6. $\pi \leftarrow$ Insertion_LS(π);
7. **if** $C_\pi < C_{\pi^*}$ **then**
8. $pool \leftarrow \emptyset$, $F_{pool} \leftarrow \emptyset$, $F_{\pi^*} \leftarrow F_\pi$;
   endif
9. **if** $\pi \notin pool$ **then**
10. Add π to *pool*, add F_π to F_{pool};
    endif
11. **if** $|pool| > pool\_size$ **then**
12. Delete the worst solution in *pool*, and the corresponding objective matrix in F_{pool};
    endif
13. $\pi \leftarrow$ Perturbation();

end

14. Output π^* and stop.

Figure 3: Pseudo code of the proposed SAILS.

work, the latter is applied on line 5 in order to more easily compare the SAILS with other algorithms.

In considering insertion local search, the number of solutions to be evaluated is at least $n \times (n-1)$ and the time complexity for evaluating one solution is $O(nm)$, so the time complexity of the local search is at least $O(n^3m)$. In literature, the CPU time limitation is often set to ρnm milliseconds, where ρ is a constant (Tasgetiren et al., 2007; Ruiz and Stützle, 2007; Xu et al., 2011; Costa et al., 2012b; Costa et al., 2012a; Pan and Ruiz, 2012). This setting has a shortcoming in that smaller instances can run with more CPU time relative to larger instances. For example, an instance with 20 jobs and 20 machines may have $1000 \times 20 \times (20-1)$ solutions evaluated in $\rho \times 20 \times 20$ milliseconds CPU time, i.e. checking all the jobs 1000 passes; while an instance with 500 jobs and 20 machines still has the same number solutions evaluated in $\rho \times 500 \times 20$ milliseconds CPU time, i.e. checking all the jobs only about 1.52 passes. In order to avoid this unfair, the CPU time limitation is set to $\rho n^3 m$ milliseconds. In this work, ρ is set to 0.004, 0.012 and 0.02, respectively.

4 COMPUTATIONAL RESULTS

In this section, the proposed algorithms are evaluated. Firstly, two parameters of the SAILS are tuned, then the SAILS is compared with state of the art algorithms and shown the effective of the newly proposed perturbation method.

The benchmark instances used for analysis are

Table 2: Tuning the parameters of the SAILS.

temperature T	value of k			
	1	2	3	4
3	0.306	0.298	0.294	0.314
4	0.309	0.292	0.299	0.318
5	0.318	0.309	0.312	0.292

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 experiment, five independent runs are performed for the instances with less than 500 jobs. The ten instances with 500 jobs and 20 machines are run only once as they are considerably more time consuming. For example, for the terminal criterion $0.02n^3m$ milliseconds CPU time, about 13.9 hours are required for just one run.

The performance of the algorithms are tested using the relative percentage deviation (RPD), which is calculated as:

$$RPD = (C - C_{best})/C_{best} \times 100 \qquad (7)$$

where C is the result found by the algorithm being evaluated and C_{best} is the best result provided by Pan and Ruiz (Pan and Ruiz, 2012), found as the best solution among their four proposed algorithms and 11 other state of the art works.

Each algorithm is implemented in C++, running on three similar PCs, each with an Intel Core2 Duo processor (2.99 GHz) and 2GB main memory. Though each computer has two processors, only one is used in the experiments, as no parallel programming technique has been applied.

4.1 Tuning of the SAILS

There are two parameters in the proposed SAILS, the first one is k, used in Eq. (5); the other is the temperature T, used in the self-adaptive perturbation (see Fig. 2). The k is set to 1, 2, 3 and 4, respectively, and the T is set to 3, 4 and 5, respectively. So there are 12 combinations in total. The terminal criterion is set to $0.02 \times mn^3$. As it is too time consuming for the 500 jobs instances, the SAILS is only run on the first 110 instances. The overall averaged RPDs (ARPD) for these 12 combinations are listed in Table 2. From this table, it can be seen that the results are quite similar for each pair of T and k. However, the results are generally better with $k = 2$. As the performance is one of the best cases with $k = 2$ and $t = 4$, we choose them in the following experiments.

Table 3: Comparison in ARPD for MRSILS, PR-ILS, PR-IGA and SAILS ($0.004n^3m$ ms CPU time).

n|m	MRSILS	PR-ILS	PR-IGA	SAILS
20|5	0.007	0	0.016	0.007
20|10	0.002	0	0	0
20|20	0	0	0	0
50|5	0.476	0.558	0.485	0.501
50|10	0.575	0.677	0.630	0.584
50|20	0.534	0.610	0.589	0.393
100|5	0.824	0.888	0.819	0.834
100|10	1.032	0.941	1.127	1.022
100|20	1.076	1.043	1.171	0.991
200|10	0.740	0.754	0.667	0.750
200|20	0.393	0.283	0.296	0.293
500|20	0.121	0.020	0.063	0.149
Avg.	0.482	0.481	0.488	0.460

4.2 Evaluation of the SAILS

The values for the average RPD (ARPD) of all 120 instances are presented in Tables 3 - 5 for each combination of n and m. From these tables, it can be seen that SAILS generally performs the best. And with the prolonging of CPU time, the superiority of the SAILS increases (See Fig. 4). The reason for this is that the self-adaptive strategy will be applied more times in the search with prolonged CPU time, and then resulting significantly better performance for the SAILS. This also shows that the proposed strategy is effective, particularly with longer CPU time.

This phenomenon can also be observed on large instances. With the 500 job instances, when the CPU time limitation is set to $0.004 \times n^3m$, SAILS performs the worst. When the CPU time limitation is set to $0.012 \times n^3m$, SAILS surpasses MRSILS and PR-IGA, but is still worse than PR-ILS. With the CPU time limitation set to $0.020 \times n^3m$, SAILS further outperforms MRSILS and PR-IGA, and is quite similar to PR-ILS.

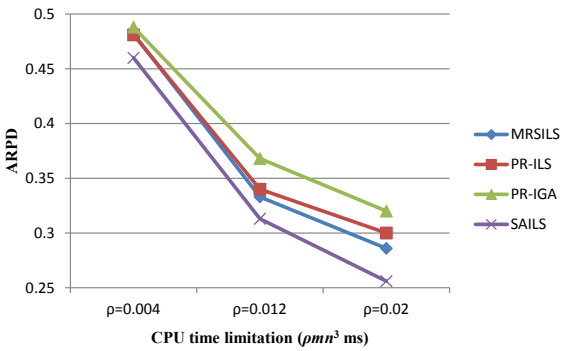

Figure 4: Comparison results for the SAILS with the MRSILS, PR-ILS and PR-IGA with different CPU time settings.

Table 4: Comparison in ARPD for MRSILS, PR-ILS, PR-IGA, SAILS ($0.012n^3m$ ms CPU time).

$n\|m$	MRSILS	PR-ILS	PR-IGA	SAILS
20|5	0.007	0	0.007	0.007
20|10	0	0	0	0
20|20	0	0	0	0
50|5	0.345	0.406	0.279	0.363
50|10	0.398	0.494	0.572	0.462
50|20	0.388	0.431	0.537	0.303
100|5	0.646	0.703	0.678	0.650
100|10	0.766	0.739	0.897	0.751
100|20	0.841	0.836	0.853	0.738
200|10	0.547	0.597	0.527	0.536
200|20	0.083	-0.016	0.097	0.007
500|20	-0.027	-0.110	-0.034	-0.057
Avg	0.333	0.340	0.368	0.313

Table 5: Comparison in ARPD for MRSILS, PR-ILS, PR-IGA, SAILS ($0.02n^3m$ ms CPU time).

$n\|m$	MRSILS	PR-ILS	PR-IGA	SAILS
20|5	0.007	0	0.007	0.007
20|10	0	0	0	0
20|20	0	0	0	0
50|5	0.296	0.332	0.262	0.290
50|10	0.357	0.474	0.527	0.402
50|20	0.387	0.416	0.480	0.274
100|5	0.602	0.648	0.590	0.557
100|10	0.699	0.697	0.793	0.659
100|20	0.701	0.740	0.761	0.612
200|10	0.473	0.521	0.475	0.460
200|20	-0.002	-0.063	0.027	-0.046
500|20	-0.089	-0.165	-0.081	-0.144
Avg	0.286	0.300	0.320	0.256

5 CONCLUSIONS

Iterated local search algorithms are powerful for solving the PFSP minimizing total flow time, with the perturbation method playing an important role. The MRSILS algorithm by Dong et al. (Dong et al., 2013) is a state of the art ILS algorithm. However, one shortcoming is that the best current solution cannot be improved in many local search runs during the search process. Further, the perturbation method only moves a randomly selected job to another randomly selected position, without any bias, such that it is difficult to escape from a "deep" local optimum.

In order to overcome this shortcoming, a self-adaptive perturbation method is proposed in this paper. In this method, the search status is evaluated by calculating the average objective value of a sample of the neighborhoods around the local optimum. The greater the difference between the average objective value and the best current objective value, the higher the probability that the perturbation strength is increased and a randomly selected job is moved to a position where worse solutions can be generated. The SAILS algorithm is proposed based on the above analysis. Experimental results on benchmark instances show that SAILS works quite well, especially for long CPU times. The proposed methodology developed here may potentially be applied to other problems, as escaping local minimums with local search is a difficulty for many combinatorial optimization problems.

ACKNOWLEDGEMENTS

This work is supported by The Fundamental Research Funds for the Central Universities of China (Project Ref. 2014JBM034, Beijing Jiaotong University).

REFERENCES

Costa, W., Goldbarg, M., and Goldbard, E. (2012a). Hybridizing VNS and path-relinking on a particle swarm framework to minimize total flowtime. *Expert Systems with Applications*, 39:13118–13126.

Costa, W., Goldbarg, M., and Goldbard, E. (2012b). New VNS heuristic for total flowtime flowshop scheduling problem. *Expert Systems with Applications*, 39:8149–8161.

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 & 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 & 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.

Nikolaev, A. G. and Jacobson, S. H. (2010). *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, chapter Simulated Annealing, pages 1–39. 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. (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., Liang, Y.-C., Sevkli, M., and Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, 177:1930–1947.

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.

Xu, X., Xu, Z., and Gu, X. (2011). An asynchronous genetic local search algorithm for the permutation flowshop scheduling problem with total flowtime minimization. *Expert Systems with Applications*, 38:7970–7979.

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.