Iterated Local Search Heuristics for Minimizing Total Completion Time in Permutation and Non-Permutation Flow Shops

Alexander J. Benavides, Marcus Ritt
Instituto de Informática
Universidade Federal do Rio Grande do Sul, Brazil.

Abstract

We study the improvement of non-permutation over permutation schedules in flow shops when minimizing the total completion time. We solve both problems by a two-phase heuristic. The first phase uses an iterated local search to find a good permutation schedule. The second phase explores non-permutation schedules using an effective insertion neighborhood, that permits to anticipate or delay a job when passing from one machine to the next. In computational experiments we show that both phases yield state-of-the-art results. We find that allowing non-permutation schedules can reduce the total completion considerably with a moderate extra effort, and without increasing the buffer size needed during processing. We conclude that non-permutation schedules can be viable alternative to permutation schedules in flow shops.

1 Introduction

In a flow shop we have to schedule jobs $J_1, \ldots, J_n$ on machines $M_1, \ldots, M_m$. Each job has to be processed on all machines in the given machine order. The processing of job J_j on machine M_i is called an *operation*. It takes time p_{ij} and cannot be preempted. At any instant, each job can be processed on at most one machine, and no machine can process more than one job. In a *permutation schedule* all machines process the jobs in the same order, in a *non-permutation schedule* the processing order may be different on some machines.

The most common objective functions in flow shops are to minimize the makespan $C_{\max} = \max C_i$ and the total completion time $\sum C_j$ of the schedule, for completion times C_j of the jobs on the last machine. In this paper we focus on the second objective. For release times $r_j = 0$, minimizing the total completion time is equivalent to minimizing the total flowtime $\sum F_j$[1]. Permutation and non-permutation variants of this problem are also denoted by $F \mid \mathrm{prmu} \mid \sum C_j$ and $F \mid\mid \sum C_j$ (Graham et al. 1979).

For regular measures, which include makespan, and total completion time or flowtime, there is always an optimal schedule with the same job processing order on the first two machines (Conway, Maxwell, and Miller 1967). Thus, for $m \leq 2$ the optimal permutation and non-permutation solutions of the flow shop scheduling problem (FSSP) have the same value. For three or more machines optimal non-permutation schedules can be shorter than optimal permutation schedules, as shown in the example in Figure 1. Minimizing flowtime is strongly NP-hard for $m \geq 2$ (Garey, Johnson, and Sethi 1976). (Gonzalez and Sahni 1978) have shown that no busy schedule (i.e. a schedule where in each instant at least one machine is processing some job) has a flowtime more than n times of the optimum, and that processing jobs in non-decreasing total processing time $p_j = \sum_i p_{ij}$ yields an m-approximation. The best known approximation ratio in the permutation case is $O(\sqrt{\min\{m, n\}})$ (Nagarajan and Sviridenko 2009).

In this paper, we are interested in the difference between permutation and non-permutation schedules, and the difficulty of finding good non-permutation schedules. There are $n!^{m-1}$ candidates for optimal non-permutation schedules, compared to only $n!$ permutation schedules. Non-permutation schedules for flow shops have been mostly neglected in the literature, since they are considered to be as difficult to find as job shop schedules, whereas permutation schedules of good quality can be found much easier. However, to the extent that state-of-the-art methods for permutation schedules approach the optimal values for permutation schedules, improvements by non-permutation schedules may be interesting. (Liao, Liao, and Tseng 2006), for example, report average improvements of up to 0.94% for instances with 10 to 50 jobs and 5 to 15 machines.

To find good non-permutation schedules one can try to exploit the observation that most combinations of job processing orders on subsequent machines are very unlikely to lead to better solutions, e.g. when the processing order of the jobs is reversed from one machine to the next. Therefore the challenge is to reduce the search space to interesting non-permutation solutions of short total completion time.

The remainder of this paper is organized as follows. We discuss related work in Section 1 and present in Section 2 our two-phase local search heuristic. We present a detailed computational analysis in Section 3 and conclude in Section 4.

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1]Articles about minimizing flowtime usually suppose that all jobs are released at time 0.

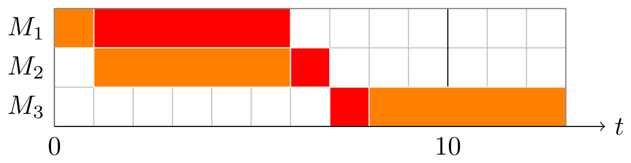

Figure 1: Gantt chart of the optimal non-permutation schedule of total completion time 21 of a three-machine two-job flow shop. The best permutation schedule has total completion time of 23 (Rinnooy Kan 1976).

Related work

(Pan and Ruiz 2013) surveyed simple and composite heuristics for minimizing total completion time in permutation flow shop, and (Pan and Ruiz 2012) compared four iterated local search methods to twelve state-of-the-art metaheuristics, including the estimation of distribution algorithm and the variable neighborhood search of (Jarboui, Eddaly, and Siarry 2009), four hybrid genetic algorithms of (Tseng and Lin 2009), (Tseng and Lin 2010), (Zhang, Li, and Wang 2009), (Xu, Xu, and Gu 2011), the discrete differential evolution algorithm of (Zhang and Li 2011), and the stochastic local search of (Dubois-Lacoste, López-Ibáñez, and Stützle 2010). Recently, (Dong et al. 2013) proposed the multi-restart iterated local search MRSILS working with a pool of solutions, and (Zhang and Wu 2014) proposed a particle swarm optimization hybridized with simulated annealing and variable neighborhood search. All the above studies are limited to permutation schedules. Only a few authors consider non-permutation schedules. (Krone and Steiglitz 1974) propose a two-phase heuristic. The first phase improves a random permutation schedule by shifting jobs, and the second phase constructs a non-permutation schedule by selective job passing. (Liao, Liao, and Tseng 2006) propose a tabu search on an insertion neighborhood and compare it to the genetic algorithm of (Reeves 1995) on six objective functions, including total completion time. Non-permutation schedules are obtained in $m-1$ phases. In phase i the permutations on machines $1, \ldots, i+1$ are fixed, and the permutation π for machines $i+2, \ldots, m$ is optimized. (Pugazhendhi et al. 2004) observe that the improvement of non-permutation schedules over permutation schedules increases with the percentage of missing operations. They derive non-permutation schedules from permutation schedules by anticipating operations that can be executed in idle intervals of the current machine.

From the above surveys and the more recent publications iterated local search methods emerge as the most effective technique to find good permutation schedules, and the tabu search of (Liao, Liao, and Tseng 2006) seems to be the current best method for finding non-permutation schedules of short total completion time. Iterated local search methods are also among the state-of-the-art techniques for minimizing the makespan. For this reason we focus in this paper on iterated local search heuristics to find non-permutation schedules.

Algorithm 1 Iterated local search

```
Input: A solution $s$.
Output: An improved solution $s'$.
 1: function ILS($s$)
 2:     $s := \mathrm{localsearch}(s)$
 3:     repeat
 4:         perturb the current solution $s$ to obtain $s'$
 5:         $s' := \mathrm{localsearch}(s')$
 6:         if $\mathrm{accept}(s, s')$ then
 7:             $s := s'$
 8:         end if
 9:     until some stopping criterion is satisfied
10:     return the best solution $s^*$ found during the search
11: end function
```

2 Iterated local search heuristics for minimizing total completion time

An iterated local search (ILS) starts from some initial solution and repeatedly applies a local search procedure to find a local minimum, followed by a perturbation to escape from it, until some stopping criterion is satisfied. To avoid visiting substantially worse local minima, often the next local minimum must satisfy some acceptance criterion; otherwise the search maintains the current local minimum. The general structure of an ILS is shown in Algorithm 1.

In our algorithms we use a Metropolis acceptance criterion in line 6. The new solution s' is accepted with probability

$$P[\mathrm{accept}(s, s')] = \min\{e^{-\Delta(s,s')/T}, 1\} \qquad (1)$$

for an increase in total completion time of $\Delta(s, s') = C_{\mathrm{sum}}(s') - C_{\mathrm{sum}}(s)$ and a *temperature* $T = \alpha\overline{p}n/10$, where

$$\overline{p} = \sum_{j\in[n]} \sum_{i\in[m]} p_{ij}/nm$$

is the average processing time of an operation. A similar criterion has been proposed by (Osman and Potts 1989) for minimizing the makespan in permutation flow shops by Simulated Annealing and has been successfully applied in several iterated local search algorithms. For the total completion time criterion the temperature has been adjusted to be a factor of n higher to account for the higher objective function value.

Besides the acceptance criterion, the performance of an ILS depends on the structure of the neighborhood, and the perturbation strength. A perturbation which is too weak may lead to stagnation, while a too strong perturbation can turn the algorithm into a randomized multi-start local search.

A special case of an ILS is an iterated greedy algorithm (IGA). Here, the perturbation is achieved by removing some random elements from the current solution and using a greedy algorithm to rebuild a complete solution. Since the rebuilding step is not random, an IGA which repeatedly destructs and reconstructs the solution can be effective even without a local search.

Evaluation of schedules

A permutation schedule is given by a permutation π of the index set $[n]$ of the job[2] a non-permutation solution $s = (\pi_1, \ldots, \pi_m)$ is defined by the permutation of the job indices on each of the m machines. Then $\pi_i(k)$ is the k-th job on machine M_i and its completion time is

$$C_{i,\pi_i(k)} = \max\{C_{i,\pi_i(k-1)}, C_{i-1,\pi_{i-1}(k)}\} + p_{i,\pi_i(k)}, \quad (2)$$

where $C_{i,\pi_i(0)} = 0$ and $C_{0,\pi_0(k)} = 0$. The completion time of job j is $C_j = C_{mj}$ and the total completion time is $C_{\text{sum}} = \sum_{j\in[n]} C_j$. Computing the total completion time using equation (2) needs time $\Theta(nm)$. Clearly, when the order of some operations has changed, only the modified completion times have to be updated. Moreover, (Duan et al. 2013) have observed that the completion times of the operations on the last machine depend only on the completion times of the operations on the critical path. This can be used to further reduce the number of completion times that must be updated. We use both techniques in our algorithms to compute updated total completion times.

An iterated local search for permutation schedules

The most common neighborhoods for permutation schedules are swapping adjacent jobs, swapping arbitrary pairs of jobs and shifting jobs (Dubois-Lacoste, López-Ibáñez, and Stützle 2010; Pan and Ruiz 2012; Rajendran and Ziegler 2004; Liu and Reeves 2001; Jarboui, Eddaly, and Siarry 2009; Tasgetiren et al. 2011). For a permutation schedule π, where π is a permutation of the index set $[n]$ of the jobs, an adjacent swap at position $i \in [n-1]$ exchanges jobs $\pi(i)$ and $\pi(i+1)$, a swap of positions $i, j \in [n]$ exchanges the jobs $\pi(i)$ and $\pi(j)$, and a shift of the job at position $i \in [n]$ to $j \in [n+1] \setminus \{i, i+1\}$ results in the permutation $(\pi(1), \ldots, \pi(j-1), \pi(i), \pi(j), \ldots, \pi(i-1), \pi(i+1), \ldots, \pi(n))$. The size of the adjacent swap neighborhood is $n-1$, the swap and shift neighborhoods have size $\binom{n}{2}$. For each neighbor the new total completion time has to be computed in time $O(nm)$ as described earlier.

The swap and shift neighborhoods have repeatedly been identified as the most effective ones. In our tests we found that the order in which the neighbors are examined has a small but consistent effect on solution quality and robustness. For this reason, and for definiteness, we propose to visit the shift neighborhood in a random order, and the swap neighborhood in order of increasing distance of the swapped jobs. This latter order will, in particular, explore the adjacent swap neighborhood, which is part of the complete swap neighborhood, first. The shift and swap local search are described in Algorithms 2 and 3. Our tests have also shown that both neighborhoods can complement each other, and thus we apply them alternately. Finally, we impose a repetition limit r on the number of full neighborhood searches to avoid occasional local minima which are costly to find. In our experiments below we have set the repetition limit to $r = 3$.

We perturb a local minimum by removing d random jobs from the current schedule, and inserting them again, one by one, at the position which minimizes the total completion time. If there are several insertion positions of minimal total completion time, we break ties by inserting the job at the first of them. The initial schedule is obtained by the constructive heuristic LR(n/m) of (Liu and Reeves 2001) applied to n/m initial sequences. The complete iterated local search for permutation schedules is shown in Algorithm 4.

[2] We use the notation $[n] = \{1, \ldots, n\}$.

Algorithm 2 Randomized shift local search.

```
Input: A permutation schedule $\pi$, a repetition limit $r$
Output: A permutation schedule $\pi'$ with $C_{\text{sum}}(\pi') \le C_{\text{sum}}(\pi)$.
 1: function RSLS($\pi$,$r$)
 2:     repeat at most $r$ times
 3:         for $j \in [n]$ in some random order do
 4:             let $\pi'$ be the result of the best shift of job $\pi(j)$
 5:             let $\pi := \pi'$ if $C_{\text{sum}}(\pi') < C_{\text{sum}}(\pi)$
 6:             return $\pi$, if the last $n$ shifts did not improve
 7:         end for
 8:     until $\pi$ did not improve
 9:     return $\pi$
10: end function
```

Algorithm 3 Swap local search.

```
Input: A permutation schedule $\pi$, a repetition limit $r$
Output: A permutation schedule $\pi'$ with $C_{\text{sum}}(\pi') \le C_{\text{sum}}(\pi)$.
 1: function SLS($\pi$,$r$)
 2:     $d := 1$
 3:     while $d \le n$ and up to $rn^2$ swaps do
 4:         for $j \in [n-d]$ do
 5:             swap jobs $\pi(j)$ and $\pi(j+d)$ to get $\pi'$
 6:             let $\pi := \pi'$, if $C_{\text{sum}}(\pi') < C_{\text{sum}}(\pi)$
 7:         end for
 8:         if $\pi$ improved then
 9:             $d := 1$
10:         else
11:             $d := d + 1$
12:         end if
13:     end while
14:     return $\pi$
15: end function
```

An iterated greedy algorithm for finding non-permutation schedules

Our main hypothesis for finding short non-permutation schedules is that only limited changes to the processing order between two machines need to be considered during the search. We propose an extended job insertion procedure which reflects this principle. When inserting a job at some position into a partial schedule, we allow job passing: the processing of a job may be anticipated or delayed at some intermediate machine.

Formally, let $\pi_1, \ldots, \pi_m$ be a partial non-permutation schedule, with j jobs. Inserting a job J at position $k \in [2, j+1]$ with anticipation after machine $i \in [m-1]$ inserts

Algorithm 4 Iterated local search for permutation schedules.

```
Input: A repetition limit $r$
Output: A permutation schedule $\pi$.
 1: function ILS($r$)
 2:     $\pi := \mathrm{LR}(n/m)$
 3:     $\pi := \mathrm{RSLS}(\pi)$
 4:     repeat
 5:         remove $d$ random jobs $J_1, \dots, J_d$ from $\pi$ to get $\pi'$
 6:         for $j \in [d]$ do
 7:             insert $J_j$ into $\pi'$ at the position which minimizes $C_{\mathrm{sum}}(\pi')$
 8:         end for
 9:         if current iteration is even then
10:             $\pi' := \mathrm{SLS}(\pi', r)$
11:         else
12:             $\pi' := \mathrm{RSLS}(\pi', r)$
13:         end if
14:         $\pi := \pi'$, with prob. $P[\mathrm{accept}(\pi, \pi')]$ eq. (1)
15:     until some stopping criterion is satisfied
16:     return the best permutation schedule $\pi$ found
17: end function
```

J into $\pi_1, \dots, \pi_i$ at position k, and into $\pi_{i+1}, \dots, \pi_m$ at position $k-1$. Similarly, inserting a job J at position $k \in [j]$ with delay after machine $i \in [m-1]$ insert J into $\pi_1, \dots, \pi_i$ at position k, and into $\pi_{i+1}, \dots, \pi_m$ at position $k+1$. Figure 2 gives examples of insertions with anticipation, without job passing, and with delay.

Insertion with job passing increases the number of possible insertions of a job from $O(n)$ to $O(nm)$. Thus, the total time finding the insertion position and the machine after which the job is anticipated or delayed, if any, such that the total completion is minimized is $O(n^2m^2)$. We obtain a iterated greedy algorithm for finding non-permutation schedules by repeatedly removing d random jobs, and reinserting them optimally into the schedule with job passing. If several insertions lead to the same total completion time preference is given to insertions without job passing over insertions with anticipation, followed by insertions with delay. The earliest insertion position is used as a second-level tie-breaker. As for the iterated local search, the new schedule is accepted according to the Metropolis criterion (1). The complete algorithm proceeds in two phases. In the first phase, the ILS is used to find a good permutation schedule, which serves as the initial solution for the second phase, which applies the IGA to find an improved non-permutation schedule. The complete procedure is shown in Algorithm 5.

3 Computational experiments

We report the results of two computational tests. The first compares the quality of the permutation schedules obtained by our iterated local search to state-of-the-art methods from the literature. The second test compares the quality of non-permutation schedules to permutation schedules. Additionally, we study the amount of job reordering between consecutive machines in non-permutation schedules. Finally, we assess the buffer requirements of permutation and non-permutation schedules.

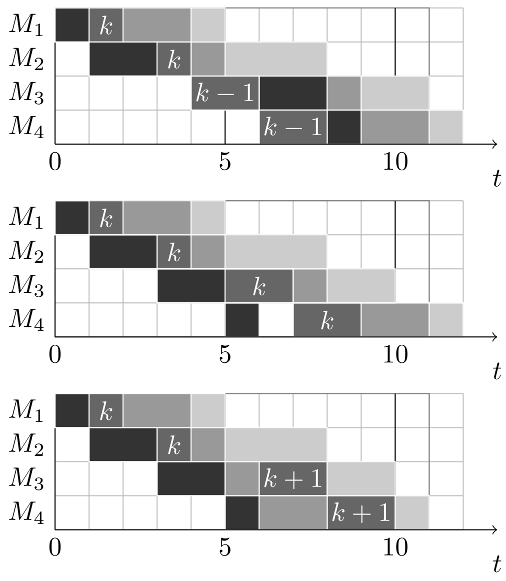

Figure 2: Top: insertion of a job at position $k = 2$ with anticipation after machine 2. Middle: insertion of a job at position $k = 2$ without job passing. Bottom: insertion of a job at position $k = 2$ with delay after machine 2.

Table 1: Sizes of the test instances.

Grp.	n	m	Grp.	n	m	Grp.	n	m
ta01	20	5	ta05	50	10	ta09	100	20
ta02	20	10	ta06	50	20	ta10	200	10
ta03	20	20	ta07	100	5	ta11	200	20
ta04	50	5	ta08	100	10	ta12	500	20

Test instances and experimental methodology

We have tested our algorithms on 120 difficult instances proposed by (Taillard 1993), which are the standard benchmark in the literature. The instances consist of 12 groups of 10 instances of the same size. Table 1 shows the number of jobs ("n") and the number of machines ("m") of each group. The processing times are uniform random numbers from the interval $[1, 99]$.

It is common to compare metaheuristics for flow shops using a time limit of τnm ms, for some constant τ. We use three different time scales in our experiments. For the ILS we adopt the shortest time scale $\tau = 30$ used in the literature. We further use a time scale of $\tau = 60$, since this is the total time needed for the two phases of the IGA. Finally, since finding non-permutation schedules is considerably harder, we report results for a longer time limit of $30nm^2$ for the IGA. We present the quality of the results as relative deviations $(C_{\mathrm{sum}} - C^*_{\mathrm{sum}})/C^*_{\mathrm{sum}}$ from the best known values C^*_{sum} reported by (Pan and Ruiz 2012). All

Algorithm 5 Iterated greedy algorithm for non-permutation schedules.

```
Input: A repetition limit $r$.
Output: A non-permutation schedule $s = (\pi_1, \dots, \pi_m)$.
 1: function IGA($r$)
 2:     $\pi := \mathrm{ILS}(r)$
 3:     let $s := (\pi, \pi, \dots, \pi)$
 4:     repeat
 5:         remove $d$ random jobs $J_1, \dots, J_d$ from each $\pi_i \in s$ to get $\pi'_i \in s'$
 6:         for $j \in [d]$ do
 7:             for all positions $k \in [n]$ do
 8:                 evaluate insertion of $J_j$ at $k$ with anticipation after machine $2, \dots, m-1$.
 9:                 evaluate insertion of $J_j$ at $k$ with delay after machine $2, \dots, m-1$.
10:                 evaluate insertion of $J_j$ at $k$.
11:             end for
12:             apply the best insertion of $J_j$ to $s'$
13:         end for
14:         $s := s'$, with probability $P[\mathrm{accept}(s, s')]$ eq. (1)
15:     until some stopping criterion is satisfied
16:     return the best permutation schedule $\pi$ found
17: end function
```

reported values are averages of 10 replications.

Our algorithms have been implemented in C++, compiled with g++ version 4.7.3 and run on an PC with an eight-core AMD FX-8150 processor running at 1.4 GHz and with 32 GB of main memory, using only one core in each execution. The detailed results reported in the tables below are available online at `http://www.inf.ufrgs.br/algopt/npcsum`.

Calibration of parameters

Both local search algorithms, ILS and IGA, depend on two parameters: the number of jobs d to remove and reinsert in a perturbation and the temperature factor α. We use the R package irace (López-Ibáñez et al. 2011) which implements an iterative F-Race (Balaprakash, Birattari, and Stützle 2007) to tune the parameters. An F-Race generates random parameters settings and compares their performance by applying the non-parametric Friedman test for comparing multiple blocks and treatments and post-hoc tests to identify the best parameter settings. The iterated version repeats this process using the best parameter settings from the previous iteration to generate new ones. We chose the three hard instances ta061, ta071, and ta081 for these tests. Based on values of the literature and some preliminary tests we have chosen parameter search ranges $\alpha \in [0.01, 0.25]$, $d \in [1, 15]$ for the iterated local search for permutation schedules, and $\alpha \in [0.0125, 2.0]$, $d \in [1, 10]$ for the iterated greedy search for non-permutation schedules. The racing algorithm was run in both cases with a budget of 2000 executions. The best parameter settings found were $\alpha = 0.2353$ and $d = 8$ for the permutation version, and $\alpha = 0.146$ and $d = 2$ for the non-permutation version.

Table 2: Comparison to the best heuristic IGA reported in (Pan and Ruiz 2012) on the instances of (Taillard 1993).

n	m	This paper		PR	
		$30nm$	$60nm$	$30nm$	$60nm$
20	5	0.00	0.00	0.00	0.00
20	10	0.00	0.00	0.00	0.00
20	20	0.00	0.00	0.00	0.00
50	5	0.23	0.17	0.51	0.46
50	10	0.33	0.27	0.75	0.70
50	20	0.41	0.36	0.75	0.67
100	5	0.61	0.52	1.03	0.91
100	10	0.83	0.69	1.43	1.23
100	20	0.98	0.88	1.49	1.35
200	10	0.67	0.57	1.08	0.93
200	20	0.60	0.41	1.00	0.82
500	20	0.36	0.30	0.52	0.45
Averages		0.42	0.35	0.71	0.63

Table 3: New upper bounds on the total completion time for permutation schedules.

Instance	C_{sum}	Instance	C_{sum}
ta099	1025946	ta115	6728404
ta103	1268383	ta118	6771654
ta109	1234115	ta119	6710587

Quality of permutation schedules

Table 2 compares our results for a time limit of $30nm$ ms and $60nm$ ms to the iterated greedy algorithm of (Pan and Ruiz 2012) (PR) with the same time limits. PR is the best of 16 heuristics evaluated by (Pan and Ruiz 2012). The table reports, for each instance group, the average relative deviation from the best known values in percent. For our algorithm we report averages of 10 replications, the values of PR are averages of 5 replications.

The machine of (Pan and Ruiz 2012) is about 10% faster than our machine, but the average relative deviations of our algorithm in $30nm$ ms is still better than that of PR in $60nm$ ms. Indeed, in 89 of the 120 instances the average total completion time of our algorithm is less than that of PR obtained in about the double of the time. Our algorithm found 6 new upper bounds on the total completion time, which are reported in Table 3.

We further compare our results to the heuristic MRSILS of (Dong et al. 2013). They report a slightly better average relative deviation compared to the heuristics DABC of (Tasgetiren et al. 2011) and hDDE of (Pan, Tasgetiren, and Liang 2008). Both DABC and hDDE are dominated by PR of (Pan and Ruiz 2012), which achieves solutions with an average relative deviation of about 0.4% less. (Dong et al. 2013) report the best values found in their experiments for instances with 50 and 100 jobs for a time limit of $400nm$ ms over 10 replications. We compare these values to the best values of

Table 4: Comparison to the results of (Dong et al. 2013)'s heuristic MRSILS: average relative deviations of the best result of 10 replications.

n	m	This paper		MRSILS
		$30nm$	$60nm$	$400nm^*$
50	5	0.10	0.07	0.09
50	10	0.13	0.09	0.17
50	20	0.18	0.17	0.21
100	5	0.44	0.39	0.45
100	10	0.58	0.41	0.53
100	20	0.61	0.59	0.55
Averages		0.34	0.29	0.34

*: about $130nm$ ms on our hardware.

our ILS for time limits of $30nm$ ms and $60nm$ ms over 10 replications in Table 4. The machine of (Dong et al. 2013) is about a factor of 3 slower than ours. After taking into account that factor, our method produces comparable results in about a quarter of the time of MRSILS, and consistently better results in half the time.

We cannot directly compare to the recent particle swarm optimization heuristic of (Zhang and Wu 2014), since neither the total completion times nor the upper bounds used to compute the relative deviations are available. Their average relative deviation in instances with up to 100 jobs is about 0.5% less than that of the particle swarm optimization of (Tasgetiren et al. 2007). (Pan, Tasgetiren, and Liang 2008) report an improvement of about 0.75% over the latter method for their IGA_{RLS} heuristic, which in turn is dominated by PR of (Pan and Ruiz 2012). Therefore, PR very likely is comparable to or dominates (Zhang and Wu 2014)'s method.

In summary, the results show that our iterated local search is competitive with the currently best methods, and thus can serve as a fair baseline in the comparison to non-permutation schedules in our next experiment.

Quality of non-permutation schedules

In our second experiment we compare the quality of permutation and non-permutation schedules. Table 5 reports average relative deviations for all 12 instance groups for the ILS for permutation schedules with a time limit of $60nm$ ms, and for the IGA for non-permutation schedules with the same time limit. We also report the results for the IGA for a longer time limit of $30nm^2$ ms. The longer time limit has been chosen to study the convergence of the search, since the non-permutation problem is considerably more difficult to solve. All values are averages of 10 replications. Negative relative deviations indicate an improvement over the current best upper bound for permutation schedules.

The comparison shows that non-permutation schedules achieve a relative deviation which is in average about 0.44% less. The improvement over permutation schedules is largest for smaller instances, and for instances with a large number of machines. Permutation schedules of the instances with 20 jobs and the instances with 20 machines improve in average by about 0.9%. The longer time limit of $30nm^2$ ms improves mostly the instances with a large number of machines, which are harder to optimize, and results in an overall improvement of 0.1%.

Table 5: Comparison of permutation (PS) and non-permutation (NPS) schedules on the instances of (Taillard 1993).

n	m	PS	NPS	
		$60nm$	$60nm$	$30nm^2$
20	5	0.00	-0.60	-0.62
20	10	0.00	-1.18	-1.28
20	20	0.00	-1.01	-1.19
50	5	0.17	0.08	0.06
50	10	0.27	-0.01	-0.10
50	20	0.36	-0.38	-0.48
100	5	0.52	0.50	0.43
100	10	0.69	0.47	0.43
100	20	0.88	0.39	0.22
200	10	0.57	0.45	0.38
200	20	0.41	0.06	-0.08
500	20	0.30	0.18	-0.01
Averages		0.35	-0.09	-0.19

Of the 120 instances, 114 non-permutation schedules have a smaller total completion time than the permutation schedules found with the same time limit of $60nm$ ms. The average total completion in 10 replications is better than the best upper bound for the permutation schedules in 49 instances, and overall 79 shorter non-permutation schedules have been found. For the longer time scale 66 non-permutation schedules are shorter in average, and 88 instances permit a shorter schedule than the best known permutation schedule.

In summary, the non-permutation schedules can improve significantly over permutation schedules, and given the current quality of heuristics for permutation schedules, it seems that additional optimization time is better invested into finding non-permutation schedules.

Amount of job reordering

Next, we investigated the amount of operations that change their processing order in the non-permutation schedules found in our second experiment. To do this, we define the job reordering index of an schedule $s = (\pi_1, \dots, \pi_m)$ as the mean number of job inversions between adjacent machines per job and number of adjacent machine pairs. This is defined mathematically as

$$\text{JRI}(s) = \frac{\sum_{i=1}^{m-1} \tau(\pi_i, \pi_{i+1})}{n(m-1)},$$

where τ is Kendall's tau distance between two permutations π and σ. Kendall's tau is the number of pairs that are in

Table 6: Comparison of average ($\bar{b}$) and maximum buffer (B) sizes of permutation (PS) and non-permutation schedules (NPS) on the instances of (Taillard 1993).

n	m	PS				NPS			
		$30nm$		$60nm$		$60nm$		$30nm^2$	
		$\bar{b}$	B	$\bar{b}$	B	$\bar{b}$	B	$\bar{b}$	B
20	5	2.0	2.0	2.0	2.0	2.0	2.1	2.0	2.0
20	10	2.2	2.2	2.2	2.2	2.2	2.3	2.1	2.4
20	20	2.7	2.7	2.7	2.7	2.4	2.8	2.4	2.9
50	5	2.8	3.3	2.7	3.2	2.7	3.4	2.8	3.3
50	10	3.1	3.8	3.1	3.9	3.0	3.5	3.1	3.8
50	20	3.9	5.2	4.0	4.9	3.8	4.6	3.8	4.8
100	5	3.9	4.6	3.9	4.6	3.9	4.7	3.7	4.4
100	10	4.4	5.6	4.3	5.0	4.3	5.2	4.4	5.3
100	20	5.7	6.9	5.6	6.5	5.5	6.3	5.6	6.8
200	10	5.2	6.3	5.3	6.2	5.2	6.4	5.0	6.1
200	20	6.5	7.9	6.4	7.6	6.4	7.7	6.4	7.7
500	20	9.3	10.4	9.1	10.1	9.2	10.3	8.9	9.9
Avg.		4.3	5.1	4.3	4.9	4.2	4.9	4.2	5.0

different order in the two permutations, namely

$$\tau(\pi,\sigma) = |\{\{i,j\} \mid \pi^{-1}(i) < \pi^{-1}(j) \wedge \sigma^{-1}(i) > \sigma^{-1}(j)\}|,$$

where $i, j \in [n]$, and π^{-1} and σ^{-1} are the inverse permutations.

We observed that the average JRI is 3.3% and the maximum JRI is 6% among 2400 non-permutation schedules, with exception of ta003 schedules that present JRI between 6% and 9%. We also counted the number of inversions that each job has on each schedule, and we observed that 83.7% of the jobs change their position only once in a schedule, 12.5% change twice, 3.2% change three times, and the remaining jobs (0.6%) change between four and sixteen times, mainly in the instances of 200 jobs or more. This confirms our hypothesis that good non-permutation schedules with a very limited amount of reordering of the operations between the machines can be found.

Buffer sizes

Finally, we evaluate the buffer requirements of permutation and non-permutation schedules. In Table 6 we report the average buffer size $\bar{b}$ and the maximum buffer size B for all permutation and non-permutation schedules found in the first two experiments. The buffer size of a single schedule is defined as the largest buffer necessary between any two machines. As in the previous experiments, the values are averages over 10 replications.

The results show that the average and maximum buffer sizes depend mainly on the number of jobs and machines of the instance, and do not vary significantly with the quality of the heuristic solution. In particular, non-permutation schedules do not lead to larger buffer sizes, and therefore can be implemented in practice without technological changes.

4 Conclusions

We have studied heuristics based on iterated local search for minimizing the total completion time in flow shops. We have proposed an ILS for permutation schedules and an IGA to find non-permutation schedules. The ILS is an evolution of current iterated local search methods and produces solutions of an average relative deviation from the current best solutions about 0.3% shorter than state-of-the-art methods on the benchmarks of (Taillard 1993). In our experiments we have found 6 new upper bounds for these instances.

When permitting non-permutation schedules the IGA can reduce the average relative deviation by another 0.44%. The gain of non-permutation schedules is particularly large for instances with a small number of jobs and a high number of machines. For the smallest instances with 20 jobs, whose current best upper bounds are presumably very close to optimal, the improvement in average relative deviation is close to 1%. In our experiments we found 88 non-permutation schedules of less total completion time than the current best upper bounds for permutation schedules. From the comparison of the results for permutation and non-permutation schedules with a time limit of $60nm\,\mathrm{ms}$, we conclude that additional computation time after $30nm\,\mathrm{ms}$ is better invested in finding non-permutation schedules.

(Liao, Liao, and Tseng 2006) found an average improvement of non-permutation over permutation schedules of 0.25% for small instances with 10 to 50 jobs and 5 to 15 machines. Our results indicate that larger improvements of about 0.44% can be achieved. The improvements of non-permutation schedules we found is comparable to the improvements of (Pugazhendhi et al. 2004) for instances with 20% missing operations. Most likely the gains for instances with missing operations can be even larger.

Acknowledgements

The research in this paper was funded in part by Petrobras, and by CNPq (project 478847/2013-0). We also a grateful for the support by the National Center for Supercomputing at UFRGS.

References

Balaprakash, P.; Birattari, M.; and Stützle, T. 2007. Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In Bartz-Beielstein, T.; Blesa, M. J.; Blum, C.; Naujoks, B.; Roli, A.; Rudolph, G.; and Sampels, M., eds., *HM 2007*, volume 4771 of *LNCS*, 108–122.

Conway, R. W.; Maxwell, W. L.; and Miller, L. W. 1967. *Theory of scheduling*. Addison-Wesley.

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. *Comput. Oper. Res.* 40:627–632.

Duan, J.; Yang, Y.; Gao, K.; Li, J.; and Pan, Q. 2013. A speed-up method for calculating total flowtime in permutation flow shop scheduling problem. In *25th Chinese Control and Decision Conference (CCDC)*, 2755–2758.

Dubois-Lacoste, J.; López-Ibáñez, M.; and Stützle, T. 2010. A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. Technical Report TR/IRIDIA/2010-019, IRIDIA, Université libre de Bruxelles.

Garey, M. R.; Johnson, D. S.; and Sethi, R. 1976. The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.* 1(2):117–129.

Gonzalez, T., and Sahni, S. 1978. Flowshop and jobshop schedules: complexity and approximation. *Oper. Res.* 26:36–52.

Graham, R. L.; Lawler, E. L.; Lenstra, J. K.; and Kan, A. H. G. R. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* 5:287–326.

Jarboui, B.; Eddaly, M.; and Siarry, P. 2009. An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. *Comput. Oper. Res.* 2638–2646.

Krone, M. J., and Steiglitz, K. 1974. Heuristic-programming solution of a flowshop-scheduling problem. *Oper. Res.* 22(3):629–638.

Liao, C. J.; Liao, L. M.; and Tseng, C. T. 2006. A performance evaluation of permutation vs. non-permutation schedules in a flowshop. *Int. J. Prod. Res.* 44(20):4297–4309.

Liu, J., and Reeves, C. R. 2001. Constructive and composite heuristics to the $P \mid\mid \sum C_i$ scheduling problem. *Eur. J. Oper. Res.* 132:439–452.

López-Ibáñez, M.; Dubois-Lacoste, J.; Stützle, T.; and Birattari, M. 2011. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université libre de Bruxelles.

Nagarajan, V., and Sviridenko, M. 2009. Tight bounds for permutation flow shop scheduling. *Math. Oper. Res.* 34(2):417–427.

Osman, I., and Potts, C. 1989. Simulated annealing for permutation flow shop scheduling. *Omega* 17(6):551–557.

Pan, Q.-K., and Ruiz, R. 2012. Local search methods for the flowshop scheduling problem with flowtime minimization. *Eur. J. Oper. Res.* 222:31–43.

Pan, Q.-K., and Ruiz, R. 2013. A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Comput. Oper. Res.* 40:117–128.

Pan, Q.-K.; Tasgetiren, M. F.; and Liang, Y.-C. 2008. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Comput. Ind. Eng.* 55(4):795–816.

Pugazhendhi, S.; Thiagarajan, S.; Rajendran, C.; and Anantharaman, N. 2004. Relative performance evaluation of permutation and non-permutation schedules in flowline-based manufacturing system with flowtime objective. *Int. J. Adv. Manuf. Technol.* 23:820–830.

Rajendran, C., and Ziegler, H. 2004. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *Eur. J. Oper. Res.* 155(2):426–438.

Reeves, C. R. 1995. A genetic algorithm for flowshop sequencing. *Comput. Oper. Res.* 22(1):5–13.

Rinnooy Kan, A. H. G. 1976. *Machine scheduling problems – Classification, complexity and computations*. Martinus Nijhoff.

Taillard, E. 1993. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* 64(2):278–285.

Tasgetiren, M.; Liang, Y.; Sevkli, M.; and Gencyilmaz, G. 2007. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *Eur. J. Oper. Res.* 177(3):1930–1947.

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

Tseng, L.-Y., and Lin, Y.-T. 2009. A hybrid genetic local search for the permutation flowshop scheduling problem. *Eur. J. Oper. Res.* 198(1):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. *Int. J. Prod. Econ.* 127(1):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. *Exp. Syst. Appl.* 38(7):7970–7979.

Zhang, Y., and Li, X. 2011. Estimation of distribution algorithm for permutation flow shops with total flowtime minimization. *Comput. Ind. Eng.* 60(4):706–718.

Zhang, L., and Wu, J. 2014. A PSO-based hybrid metaheuristic for permutation flowshop scheduling problems. *The Scientific World Journal*.

Zhang, Y.; Li, X.; and Wang, Q. 2009. Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. *Eur. J. Oper. Res.* 196:869–876.