Applying Iterated Local Search to the Permutation Flow Shop Problem

Thomas Stützle[*]

Darmstadt, University of Technology

Department of Computer Science, Intellectics Group

Alexanderstr. 10, D-64283 Darmstadt

Abstract

Iterated local search (ILS) is a general and powerful metaheuristic which provides an easily implementable way of improving the performance of local search algorithms. In this article we consider the application of ILS to the permutation flow shop problem (FSP), a strongly studied problem in machine scheduling. We experimentally investigate the effect of specific implementation choices in our ILS algorithm and analyze its performance. Computational results show that our ILS approach compares very favorably to other approaches proposed for the FSP and is, despite its simplicity, even able to find new best solutions for some benchmark instances which have already been attacked by many other algorithms.

Keywords: Metaheuristics, Iterated local search, permutation flow shop problem

1 Introduction

In the flow shop problem, each of n jobs $1, \ldots, n$ has to be processed on m machines $1, \ldots, m$ in this order. The processing time of job i on machine j is t_{ij} where the t_{ij} are fixed and nonnegative. At any time, each job can be processed on at most one machine, and each machine can process at most one job. The jobs are available for processing at time 0 and the processing of a job may not be interrupted. Here, we concentrate on the permutation flow shop problem (FSP), where the job order is the same on every

[*]Currently on leave to IRIDIA, Université Libre de Bruxelles

machine. The objective is to find a job sequence π that minimizes the completion time (called makespan) of the last job.

As the problem is $\mathcal{NP}$-hard in general [5], one has to rely on approximation methods to tackle large instances. Therefore, many algorithms have been proposed to find near optimal schedules in reasonable time. These algorithms can be classified as either constructive or based on local search. Constructive methods generate job sequences from scratch without allowing decisions to be reversed. Several constructive heuristics are described in [2, 4, 18, 11]. Local search for the FSP starts from some initial sequence and repeatedly try to improve the current sequence by local changes. If in the neighborhood of the current sequence π a better sequence π' is found, it replaces π and the local search is continued. The simplest local search algorithm, iterated descent, repeatedly applies these steps until no better sequence can be found in the neighborhood and therefore stops at the first local minimum encountered. To increase the performance of local search algorithms, the application of metaheuristics like simulated annealing [21, 20, 9], tabu search [19, 22, 26, 30], genetic algorithms [1, 23, 24], and ant colony optimization [25] has been considered for the FSP.

In this article we focus on the application of iterated local search (ILS) [16, 15, 17] to the FSP. Iterated local search (ILS) is a very simple and powerful metaheuristic which consists in repeatedly applying a local search algorithm to solutions obtained by small modifications to one of the previously visited locally optimal solutions. The simplicity of ILS stems from the fact that typically only a few lines of code have to be added to an already existing local search algorithm; the power of ILS is witnessed by the fact that ILS is among the best performing approximation algorithms for the well known traveling salesman problem [15, 10] and has been shown to be very competitive on other problems like graph partitioning [14], job-shop scheduling [12, 13], and the total weighted tardiness problem [3].

Our ILS algorithm for the FSP is based on a straightforward local search implementation. In our experimental analysis of the proposed ILS approach we focus on two particularly important aspects: from which locally optimal solution should the search be continued and how strong should be the solution modifications. We use the results of this analysis to design our final ILS algorithm and computational results obtained on a large set of widely used benchmark problems show that, despite the simplicity of the approach, competitive results to the best known algorithms for the FSP can be obtained.

The paper is structured as follows. First, we give a general algorithmic outline of ILS and discuss other algorithmic approaches falling into this category. In Section 3 we present the application of ILS to the FSP and discuss aspects of the generic choices of the operators as applied to the FSP. The experimental results are presented in Section 4 and we give some final conclusions in Section 5.

2 Iterated Local Search

Iterated local search [16, 15, 17] is a simple and generally applicable metaheuristic that iteratively applies local search to modifications of the current search point. To apply an ILS algorithm to a given problem, four components have to be specified. These are mechanism to generate an initial solution, a procedure Modify, that modifies the current solution s leading to some intermediate solution s', a procedure LocalSearch that takes s' to a local minimum s'', and an AcceptanceCriterion that decides to which solution the next time Modify is applied. An algorithmic scheme for ILS is given in Figure 1. ILS differs from other methods like simulated annealing and tabu search in the fact that it does not follow one trajectory in the search space but solution modifications which correspond to jumps in the search space are applied to allow to leave local minima.

```
procedure Iterated Local Search
    generate initial solution $s_0$
    $s$ = LocalSearch($s_0$)
    repeat
        $s'$ = Modify($s$, history)
        $s''$ = LocalSearch($s'$)
        $s$ = AcceptanceCriterion($s$, $s''$, history)
    until termination condition met
end
```

Figure 1: Pseudocode of an iterated local search procedure (ILS)

In principle, any local search algorithm can be used, but the performance of the ILS algorithm with respect to solution quality and computation speed depends strongly on the one chosen. Like in our ILS approach to the FSP, very often an iterated descent algorithm is taken. Yet, it is also possible to apply more sophisticated local search algorithms like tabu search algorithms.

The modification mechanism (we will also refer to it as *kick-move* in the following) should be chosen *strong enough* to allow to leave the current local minimum and to enable the local search to find new, possibly better local minima. At the same time, the modification should be *weak enough* to keep enough characteristics of the current local minimum. Small modifications allow the local search algorithm to run fast, because it will require only few steps to reach the next local optimum. If the modification of the current local optimum is too large, the effect would be similar to starting from a new, randomly generated solution.

The acceptance criterion is used to decide from which solution the search is continued by applying the next kick-move. One important aspect of the acceptance criterion is to introduce a bias between intensification and diversification of the search. Intensification of the search around the best found solution is achieved, for example, by applying the kick-move always to the best found solution. Diversification may be achieved, in the extreme case, by accepting every new local optimum s'', this would be similar to a random walk over the local optima as the objective function value is not taken into account. We will show that the particular choice of the acceptance criterion is critical for the performance of the ILS algorithm.

3 Application of Iterated Local Search to the FSP

To apply ILS to the FSP the three generic operators Modify, LocalSearch, and AcceptanceCriterion have to be defined and an initial solution has to be generated. In the following we detail the choices for the three operators used for the ILS application to the FSP.

3.1 Initial Solution

We use the NEH heuristic [18] to construct an initial solution. The NEH heuristic appears to be the best performing construction heuristic for the permutation Flow Shop Problem on a wide variety of problem instances [28, 11]. For the NEH we use the efficient implementation due to Taillard [26] such that NEH can be run in $\mathcal{O}(n^2m)$. Obviously, the ILS algorithm could be started from randomly chosen one. Yet, for short run times on large instances we observed that by using the solution generated by the NEH-heuristic a better solution quality could be obtained.

3.2 Choice of LocalSearch

For the FSP we considered local search algorithms based on the following neighborhood definitions which are proposed in literature. (i) swaps of two neighboring jobs at position i and $i+1$ (*swap-moves*), (*ii*) interchanges of jobs at ith and jth position (*interchange-moves*), (*iii*) remove job at ith position and insert it in the jth position (*insertion-moves*).

Local search based on swap-moves is very fast, yet the solution quality is very low and therefore we did not consider it further. In [26, 21] it was shown that the neighborhood based on insertion-moves can be evaluated more efficiently than the one based on interchange-moves and gives at least the same solution quality. Therefore we use a local search procedure based on the insertion neighborhood. The moves are defined as follows: Let (i, j) be a pair of positions. The new permutation π' is obtained by removing the job $\pi(i)$ at position i and inserting it at position j. If $i < j$ we obtain $\pi' = (\pi(1), \ldots \pi(i-1), \pi(i+1), , \ldots \pi(j), \pi(i), \pi(j+1), \ldots, \pi(n))$ and if $i > j$ we get $\pi' = (\pi(1), \ldots \pi(j-1), \pi(i), \pi(j), \ldots \pi(i-1), \pi(i+1), \ldots, \pi(n))$. The size of the neighborhood is $(n-1)^2$; using the fast neighborhood evaluation of [26], the set of possible moves can be examined in $O(n^2 m)$.

For large FSP instances the computation time for the local search still grows fast and therefore we use a modified first-improvement strategy which resulted in smaller run-times. For a randomly chosen position i we examine all possibilities for inserting job $\pi(i)$ and if an improved schedule is found, we perform the best insertion-move found during the neighborhood scan for job $\pi(i)$.

3.3 Choice of Modify

For Modify we consider simple modifications that cannot be reversed directly by the local search algorithm. We proceed by applying a number of swap-moves or at least one interchange-move. In a swap-move $\pi = (\pi(1), \ldots, \pi(i), \pi(i+1), \ldots, \pi(n))$ is modified to $\pi' = (\pi(1), \ldots, \pi(i+1)\ \pi(i), \ldots, \pi(n))$; in an interchange-move $\pi = (\pi(1), \ldots, \pi(i), \ldots, \pi(j), \ldots, \pi(n))$ is modified to $\pi' = (\pi(1)\ , \ldots, \pi(j), \ldots, \pi(i),\ \ldots, \pi(n))$. Initially, we tried also moves by simultaneously exchanging jobs at three or four randomly chosen positions, but these moves did not improve the results. Experimentally, we found that rather small modifications are sufficient to yield very good performance. Yet, the appropriate kick-strength seems to be instance dependent (see the study of an appropriate kick-move strength in Section 4.1.2). For the experimental results in Section 4 the kick-move is composed of two swap-moves at randomly chosen positions $i = 1, \ldots, n-1$ and one interchange-move with $|i - j| \leq \max\{n/5, 30\}$.

The restriction on the distance of two jobs in an interchange-move is imposed to avoid a strong disruption of an already good solution.

3.4 Choice of AcceptanceCriterion

While in most ILS applications [7, 10, 15] only better local minima are accepted, the importance of the choice of the acceptance criterion has largely been neglected. An exception is the study of ILS for the Job Shop scheduling problem in [12] and the fact that in [16] it is mentioned that for one traveling salesman problem instance, using a Simulated Annealing type acceptance criterion improves the computational results.

We considered several choices for the acceptance criterion. As the standard one we use *Better*(s, s''), which returns the better of both solutions. Using this acceptance criterion the kick-move will be only applied to the best found solution during the algorithms run. Yet, with this choice the algorithm may get trapped in certain regions of the search space. Therefore, acceptance criteria which allow moves to worse solutions may be preferable to avoid such situations. In the extreme case, one could accept every new local optimum irrespective of its solution quality. This choice will be denoted as *RW*(s, s'') (for Random Walk). Between these two extreme choices for the acceptance criterion are those accepting worse solutions with a certain probability or making use of the search history. In a Simulated Annealing type acceptance criterion, denoted as *LSMC*(s, s'') (for Large-Step Markov Chains [16]), s'' is accepted with a probability of $\exp\{(C_{\max}(s) - C_{\max}(s''))/T\}$ if s'' is worse than s; if s'' is better than s it is always chosen. T is a parameter called temperature and it is lowered during the run of the algorithm. In fact, the temperature schedule we use is a non-monotonic temperature schedule like proposed in [8] for Simulated Annealing or for Tabu thresholding [6]. We increase the temperature to an intermediate level if the minimal temperature is reached. A disadvantage of using *LSMC*(s, s'') is that a schedule for reducing the parameter T must be given. A simpler idea is to use a constant temperature T_c; we refer to this criterion as *ConstTemp*(s, s''). Similar to non-monotonic temperature schedules is an acceptance criterion *Reheating*(s, s''). In *Reheating*(s, s'') only better solutions are accepted until no improved solution is found for a number of iterations, here $2 \cdot n$. Then, the probability of accepting worse solutions is increased for a number of iterations (done by using a simulated annealing type acceptance criterion with high temperature T) to diversify the search. After such a diversification phase again only better quality solutions are accepted.

4 Experimental Results

In this section we first investigate the appropriate choice of the acceptance criterion and an appropriate kick-move strength for the ILS algorithm. Based on the results of this analysis we derive our final ILS algorithm which is then compared to the best local search algorithms known for the FSP. The experiments use a large set of benchmark instances, originally proposed by Taillard in [27]. All instances are available from ORLIB at `http://mscmga.ms.ic.ac.uk/info.html`. There are in total 120 instances which differ in the number n (chosen from the set $\{20, 50, 100, 200, 500\}$) of jobs and the number m (chosen from $\{5, 10, 20\}$) of machines. For each problem size ten instances are available; they are generated by choosing the duration of every operation randomly according to a uniform distribution from the integer numbers between 0 and 99. Of these instances, those with 20 machines are the hardest to solve to optimality. In fact, from the 20 machines instances only those with 20 jobs have been solved to optimality, while for all 5 or 10 machine instances the exact optimum is known. Therefore, we focus in most experiments on the instances with 20 machines. Note that the best known solutions for the larger instances with $n \geq 50$ improved considerably over the solutions presented in [27]; therefore, if one wants to compare the computational results with those of other articles, one has to adjust for this fact.

4.1 Analysis of the ILS algorithm

To reduce the computation burden of the experimental analysis of the ILS algorithm, we first study the influence of the different acceptance criteria proposed in Section 3.4 and then the effect of different kick-move strengths using the best performing acceptance criterion. The study on the acceptance criteria was done using a kick-move consisting of two swap-moves and one interchange-move as specified in Section 3.3. We have run the experiments extensively on four instances with 20 machines and 20, 50, 100, 200 jobs, respectively. Similar experiments on other instances have shown similar behavior.

4.1.1 Experimental comparison of Acceptance Criteria

Here we show that the choice of the acceptance criterion may be crucial to reach peak performance for long runs of an ILS algorithm. To illustrate this point, we present in Figure 2 plots of the tradeoff between CPU-time and solution quality (given as percentage deviation from the best known solutions) for FSP instances `ta021` (20×20), `ta053` (50×20), `ta087` (100×20), and `ta105` (200×20) when using different acceptance criteria in the ILS algorithm. The plots are given for acceptance criteria *Better*(s, s''), *RW*(s, s''), *LSMC*(s, s'') and *ConstTemp*(s, s''), and *Reheating*(s, s''). Except for *Better*(s, s''), all the other acceptance criteria allow moves to worse local minima.

From the solution quality versus CPS-time tradeoff given in Figure 2 on instances `ta021` and `ta053` it can be oberved that the use of acceptance criterion *RW*(s, s'') performs worst and that a bias towards better local minima is necessary to yield best ILS performance. (We did not apply the ILS algorithm using *RW*(s, s'') to the 100 and 200 job instances.) Yet, on all the instances the best performance is obtained if sometimes worse solutions are accepted. In particular, on all instances except `ta105`, *Better*(s, s'') performs worse than accetpance criteria *ConstTemp*(s, s''), *LSMC*(s, s''), and *Reheating*(s, s''). Of these latter three acceptance criteria none is better than any of the others on all instances, yet for the following experiments we decided to use *ConstTemp*(s, s'') because it needs only one single parameter to be set and overall appears to be preferable to *Reheating*(s, s'').

4.1.2 Experimental comparison of kick-move strength

In this section we investigate the appropriate kick-move strength for the ILS algorithm using acceptance criterion *ConstTemp*(s, s''). The kick-moves we apply are composed of a number of swap-moves at random positions and a number of interchange-moves; the more moves of a particular type are applied, the larger is the disruption of the solution and, hence, the strength of the kick-move. Also, the solution disruption incurred by an interchange-move is stronger that the disruption of the swap-moves, because the latter only affects a pair of adjacent jobs. In Figure 3 we give the CPU-time versus solution quality tradeoff measured on the instances used in the previous section.

As can be observed, there is no single kick-move strength which gives best performance. Surprisingly, with increasing instance size the appropriate kick-move strength does not increase; rather it decreases with incresing instance size. Except for the smallest instance `ta021` the kick-move introducing the largest solution modification (it consists of 2 swap-moves and 2 interchange-moves and is denoted by ”S2-I2” in Figure 3), gives worst performance. Additionally, the experimental results suggest that it suffices to use kick-moves which are only composed of a number of swap-moves (applying only two swap-moves gives very good performance) to achieve best performance. Hence, we can conclude that surprisingly small and simple kick-moves suffice to achieve a very good performance of the ILS algorithm.

4.1.3 Synopsis

The experimental analysis of the proposed ILS algorithm has shown that (i) best performance is achieved if the acceptance criterion allows moves to worse solutions and in this way allows a stronger exploration of the search space and that (ii) an appropriate kick-move strength depends on the particular instance and the instance size. In the ILS algorithm used for the following comparison to the best performing algorithms proposed for the FSP, we take into account these two observations. We use the acceptance criterion *ConstTemp*(s, s'') and vary the kick-move strength during the run of the algorithm, similar as proposed in the simple variable neighborhood search [7]. In particular, we vary the kick-move strength k between some minimal and maximal limits $k_{\min}$ and $k_{\max}$. Let k be the current number of swap-moves to be applied. (We do not use interchange-moves in our final algorithm.) Then, if after the application of Modify and the subsequent local search not a better solution than the previous one is found, we set $k = k + 1$; if a better solution is found, we set $k = k_{\min}$. If we reach the upper limit $k_{\max}$ without having found an improved solution, we set k to $k_{\min}$ and repeat the same cycle.

4.2 Comparisons to other algorithms

The FSP has been attacked by several metaheuristics like simulated annealing [21, 20, 9], tabu search [19, 22, 26, 30], and genetic algorithms [1, 23, 24]. Among the simulated annealing approaches, the SAOP algorithm by Osman and Potts [21] appears to be the best performing one and at the same time is easily implementable. We first compare ILS to SAOP since both SAOP and ILS are general algorithms that do not make use of special characteristics of the FSP. Furthermore, the first ILS algorithms were proposed with the idea of embedding local search into a Simulated Annealing type algorithm [16], further motivating this comparison. Currently, the tabu search algorithm by Nowicki and Smutnicki (Tabu-NS) [19] and the sophisticated genetic algorithm (which includes elements of path relinking) by Reeves and Yamada (GA-RY) [24] are the best performing algorithms for the FSP. We additionally compare the performance of the proposed ILS algorithm to these two algorithms as well as to an earlier tabu search algorithm due to Taillard (Tabu-T) [26]. (Note that our ILS algorithm uses a local search implementation analogous to the one of Tabu-T and the much better performance of ILS when compared to Tabu-T (see Section 4.2.2) shows the power of out proposed ILS approach.) For this comparison we allow for ILS the same number of local search iterations as for the Tabu-NS algorithm. Note that the Tabu-NS algorithm is a sophisticated implementation which strongly exploits problem specific characteristics to speed-up the local search. Yet, here we are not trying to set new records in computational speed to solve the permutation FSP, but we are more interested in showing that ILS is an easily implementable way of strongly improving local search performance for the FSP.

In the following experiments we use the following parameter settings. $k_{\min}$ is set to 2 and $k_{\max}$ is set to 7. The temperature T_c is set to $2./3. \cdot \bar{p}/10$, where $\bar{p}$ is the average processing time and $\bar{p}/10$ is the initial temperature used in the SAOP algorithm.

4.2.1 Experimental comparison – SAOP

In this section we compare the performance of the ILS approach to the NEH heuristic [18], to the NEH heuristic followed by local search (NEH+ls), to SAOP, and to a multiple iterated descent algorithm starting from randomly generated solutions (MD). SAOP gives a good indication of the performance of SA on the FSP because it is still the best performing Simulated Annealing algorithms for the FSP; a modified Simulated Annealing algorithm proposed in [9] yields only minor improvements on small instances and is slightly worse on the larger ones. For SAOP the maximal number of iterations is given as $\max\{3300 \cdot \ln n + 7500 \ln m - 18500, 2000\}$ [21] and therefore we allow for ILS and MD the same computation time as needed by SAOP.

The computational results are given in Table 1. As expected, ILS performs significantly better than MD and significantly improves over the solution quality achieved by NEH or NEH with additional local search or SAOP. Surprisingly, even the MD approach gives slightly better solutions than the Simulated Annealing algorithm, contradicting computational results presented in [21]. We could verify that this fact is due to the local search implementation with the improvements suggested in [26] and the first-improvement pivoting rule used in our local search algorithm. On average, the first-improvement and the best-improvement version of the local search yield similar solution quality, but the first-improvement local search is much faster and allows to apply more often a local search.

Our (general purpose) ILS approach also performs well compared to other algorithms proposed specifically for the FSP. The path algorithm proposed by Werner [29] has been applied to the instances above, of size $n = 20, m = 10$, $n = 20, m = 20$, $n = 50, m = 10$, $n = 50, m = 20$ averaging 1.46%, 1.30%, 1.97%, and 2.76% above the best known upper bound at that time. Adjusting our results to the then best known solutions, ILS averages 0.39%, 0.27%, 0.77%, and 1.57% above those. Yet, ILS needs significantly longer times (adjusting for the differences in computer speeds in [29] and here) on the instances with 20 jobs. On the instances with 50 jobs the path algorithm takes roughly 70% of the time needed by ILS.

Table 1: Results for short runs of ILS in benchmark instances. Given is the average percentage excess over the best known solution averaged over 10 instances of each size. The maximal CPU-time for SAOP, ILS, and MD is given in seconds on a Sun Sparc 5 Workstation; the computation time for NEH and NEH+ls are significantly lower.

instances	NEH	NEH + ls	SAOP	ILS	MD	CPU-time
ta001 - ta010 (20×5)	3.300	1.687	1.061	**0.419**	0.765	0.3
ta011 - ta020 (20×10)	4.601	2.451	1.462	**0.332**	0.955	1.3
ta021 - ta030 (20×20)	3.731	2418	1.116	**0.287**	0.628	3.3
ta031 - ta040 (50×5)	0.727	0.261	0.597	**0.149**	0.255	0.9
ta041 - ta050 (50×10)	6.453	3.456	3.012	**1.470**	2.873	3.3
ta051 - ta060 (50×20)	5.971	4.658	3.533	**2.131**	3.524	9.0
ta061 - ta070 (100×5)	0.527	0.324	0.509	**0.203**	0.219	1.9
ta071 - ta080 (100×10)	2.215	1.250	1.720	**0.769**	1.287	7.0
ta081 - ta090 (100×20)	5.106	4.089	4.076	**2.269**	3.822	20.50
ta091 - ta100 (200×10)	1.257	0.917	1.478	**0.738**	0.978	16.0
ta101 - ta110 (200×20)	4.245	3.152	4.078	**2.258**	3.536	46.0

Comparing the ILS results with Tabu Search algorithms [30, 26, 22, 19], ILS appears to outperform than the approaches of [30, 26] (see next section), similar to that of Reeves [22] and worse with respect to computation time than Tabu-NS [19]. (Recall that we use the simplest form of local search in our ILS.) For the instances solved, ILS gives better solution quality on instances with up to 100 jobs than the Tabu Search of [22] and slightly worse results for the larger benchmark instances with 200 and 500 jobs (adjusting our results to the then best known solutions for the benchmark instances in [27]). For the same computation time, Tabu-NS, the best performing local search algorithm for the FSP, gives significantly better results than ILS because their local search is very fine-tuned and therefore much faster than ours. Yet, as shown in the next section, if the comparison is made using the same number of local search iterations, our ILS approach performs similarly to Tabu-NS. Nevertheless, our point here is that given any local search algorithm, ILS is easy to implement, and will cheaply do better than the corresponding local search.

4.2.2 Experimental comparison – tabu search

Here we compare the solution quality obtained by our ILS algorithm to published results of two Tabu Search algorithms, namely Tabu-NS and Tabu-T, and to the genetic algorithm GA-RY. The ILS algorithm

Table 2: Results for long runs of ILS on instances with $n = 50$, $m = 20$. 30000 Local search iterations for Tabu-NS and each of 10 runs for ILS, for Tabu-T the best result of 3 runs with 50000 local search iterations each is given. For GA-RY we give the average solution quality, averages for GA-RY are taken over 30 runs (taken from Table 1 in [24]). See the text for more details.

Instance	best-known	ILS			Tabu-NS	Tabu-T	GA-RY
		best	avg.	worst			
ta051	3855	3866	3883.5	3893	3875	3886	3880
ta052	3707	3714	3715.2	3720	3715	3733	3716
ta053	3643	3656	3665.8	3678	3668	3673	3668
ta054	3731	3736	3742.6	3756	3752	3755	3744
ta055	3614	3614	3622.8	3633	3635	3648	3636
ta056	3686	3689	3699.2	3708	3698	3719	3701
ta057	3706	3711	3720.9	3729	3716	3730	3723
ta058	3700	3714	3721.2	3739	3709	3737	3721
ta059	3743	3757	3763.5	3773	3765	3772	3769
ta060	3767	3767	3769.6	3777	3777	3791	3772

is allowed the same number of local search iterations as Tabu-NS, for Tabu-T the number of local search iterations is at least twice as much as for Tabu-NS and ILS. In [24], Reeves and Yamada also compare their genetic algorithm to Tabu-NS using the same results as we do here; therefore, we can compare their results directly to our ILS algorithm.

We run ILS 10 times on each instance with $n = 50$ and $n = 100$, and 5 times on the instances with 200 jobs. Given are the best, the average and the worst makespan obtained for each instance. For Tabu-T only the best results over several runs are given in [27]; for GA-RY we use the average solution quality given in Table 1 in [24]. We only present results for instances with $n \geq 50$ as the smaller instances were solved to optimality in almost every run. Furthermore, we only report the results for instances with $m = 20$; they are the hardest instances to solve and the only ones for which the exact optimum is still unknown.

The results in Tables 2 to 4 show that ILS gives for most instances a better solution quality – even in the worst of the runs – than Tabu-T, although for Tabu-T at least twice as many local search iterations were allowed. Therefore, we can conclude that ILS is significantly more effective on the FSP instances used for comparison than Tabu-T. When comparing ILS to Tabu-NS, both seem to give rather similar solution quality on the 50 job instances (the average performance of ILS is better on 5 instances), while for the larger instances with 100 and 200 jobs the advantage of ILS appears to increase (it gives better average performance on 8 out of 10 instances). Additionally, the best solutions found by ILS are in all except of two cases better than the solutions returned by Tabu-NS. Compared to GA-RY, ILS performs roughly similar on the 50 and 100 job instances (on the 50 job instances it achieves on 8 out of 10 instances a slightly better average performance, in the case of 100 job instances on 3 out of 10), but it seems to be preferable on the 200 job instances reaching on 9 out of 10 instances better average solution quality (on many of these instances even the worst solution obtained by the ILS algorithm is better than the average solution found by GA-RY). Hence, we can conclude that the proposed ILS approach achieves very high quality solutions, comparable or better than the best local search algorithms known for the FSP.

Concerning the run-time, an important point is that a single iteration of Tabu-NS can be performed in significantly less time than a single iteration of ILS. One complete scan of the neighborhood for ILS takes $\mathcal{O}(n^2m)$, the same as for Tabu-T. But for Tabu-NS it is reported that the neighborhood, empirically, can be scanned in $\mathcal{O}(n^{1.114}m^{1.1})$. Therefore the computation times of Tabu-NS are significantly smaller than for ILS by a factor of roughly 10 to 30. The smaller complexity of the local search in Tabu-NS is due to the use of speed-up techniques based on block properties of the FSP and a largely reduced neighborhood by restricting the possible positions for job insertions. This neighborhood reduction gives considerable speed-up for instances having many jobs and relatively few machines. The techniques to obtain the speed improvements for the local search iterations reported in [19] are quite general and could also be used for the local search employed in our ILS approach. Yet, we did not implement these speed-up techniques, as our main aim is to show that the ILS algorithms may give a high solution quality for the FSP. One might argue that the local search algorithm used in our ILS approach searches a larger neighborhood than the Tabu-NS and therefore for some instances better solutions are found. Yet, Tabu-NS finds much better solutions than Tabu-T which uses the same neighborhood size as our local search implementation in a significantly lower number of local search iterations (see Table 3 in [19]). Therefore, the reason for the good performance of our ILS is not simply the larger neighborhood scanned, but is based on the structure of the algorithm itself.

5 Conclusion

In this article we have presented a new application of iterated local search to the permutation flow shop problem. We have shown that ILS increases significantly the performance of a basic local search al-

Table 3: Results for long runs of ILS on instances with $n = 100$, $m = 20$. 15000 Local search iterations for Tabu-NS and each of 5 runs for ILS, for Tabu-T the best result of 3 runs with 10000 local search iterations each is given. For GA-RY we give the average solution quality, averages for GA-RY are taken over 30 runs (taken from Table 1 in [24]). See the text for more details.

Instance	best-known	ILS			Tabu-NS	Tabu-T	GA-RY
		best	avg.	worst			
ta081	6228	6237	6260.8	6273	6286	6330	6259
ta082	6210	6217	6235.6	6257	6241	6320	6234
ta083	6271	6300	6311.1	6323	6329	6364	6312
ta084	6269	6303	6319.8	6366	6306	6331	6303
ta085	6319	6349	6372.3	6398	6377	6405	6354
ta086	6380	6403	6417.6	6437	6437	6487	6417
ta087	6292	6298	6312.8	6330	6346	6379	6319
ta088	6423	6436	6457.5	6500	6481	6514	6466
ta089	6275	6312	6323.4	6338	6358	6386	6323
ta090	6434	6471	6480.1	6483	6465	6534	6471

gorithm for the permutation flow shop problem with only very small additional implementation effort. Despite the simplicity of the approach, the experimental results show that very high solution quality can be obtained, similar to the best known local search algorithms fine-tuned to the FSP. We expect that still better performance is achievable by using a more fine-tuned local search. The ILS approach the provides a very cheap and effective way to improve upon state of the art local search methods.

Concerning the implementation of ILS algorithms, we have shown that the choice of acceptance criterion is crucial to obtain very high solution quality; on the other side our experiments have shown that the appropriate kick-move strength may be instance dependent. While the choice of an appropriate kick-move strength is in mainly the target of the recently introduced simple variable neighborhood search, the choice of the acceptance criterion seems not to have been addressed sufficiently in earlier ILS applications. Therefore, our results also suggest that the appropriate choice of the acceptance criterion should receive more attention in future ILS applications.

Acknowledgments The author would like to thank Olivier Martin for reading an earlier draft and his valuable comments that improved the presentation of the paper.

Table 4: Results for long runs of ILS on instances with $n = 200,\ m = 20$. 10000 Local search iterations for Tabu-NS and each of 5 runs for ILS, for Tabu-T the best result of 10 runs with 4000 local search iterations is given. For GA-RY we give the average solution quality, averages for GA-RY are taken over 30 runs (taken from Table 1 in [24]). See the text for more details.

Instance	best-known	ILS			Tabu-NS	Tabu-T	GA-RY
		best	avg.	worst			
ta101	11195	11244	11266.8	11292	11294	11393	11316
ta102	11223	11276	11290.8	11303	11420	11445	11346
ta103	11337	11402	11411.0	11424	11446	11522	11458
ta104	11299	11341	11354.4	11373	11347	11461	11400
ta105	11260	11297	11310.0	11327	11311	11427	11320
ta106	11189	11237	11262.6	11277	11282	11368	11288
ta107	11386	11431	11443.3	11451	11456	11536	11455
ta108	11334	11401	11441.6	11484	11415	11544	11426
ta109	11192	11247	11266.4	11294	11343	11424	11306
ta110	11313	11385	11407.2	11435	11422	11548	11409

References

[1] C. Bierwirth and S. Stöppler. The application of a parallel genetic algorithm to $n/m/P/C_{\max}$ flow-shop problem. In G. Fandel, Th. Gulledge, and A. Jones, editors, *New Directions for Operations Research in Manufacturing*, pages 161–179. Springer Verlag, Berlin, Germany, 1992.

[2] H. Campbell, R. Dudek, and M. Smith. A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, 16(10):B–630–B–637, 1970.

[3] R. K. Congram, C. N. Potts, and S. L. Van de Velde. Dynasearch — iterative local improvement by dynamic programming: The total weighted tardiness problem. In *Talk presented at CO98, Brussels, Belgium*, 1998.

[4] D. Dannenbring. An evaluation of flow shop sequencing heuristics. *Management Science*, 23(11):1174–1182, 1977.

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

[6] F. Glover. Tabu thresholding: Improved search by nonmonotonic trajectories. *ORSA Journal on Computing*, 7(4):426–442, 1995.

[7] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer, Boston, 1999.

[8] T. C. Hu, A. B. Kahng, and C.-W. A. Tsao. Old bachelor acceptance: A new class of non-monotone threshold accepting methods. *ORSA Journal on Computing*, 7(4):417–425, 1995.

[9] H. Ishibuchi, S. Misaki, and H. Tanaka. Modified simulated annealing algorithms for the flow shop sequencing problem. *European Journal of Operational Research*, 81:388–398, 1995.

[10] D. S. Johnson and L. A. McGeoch. The travelling salesman problem: A case study in local optimization. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, England, 1997.

[11] C. Koulamas. A new constructive heuristic for the flowshop scheduling problem. *European Journal of Operational Research*, 105:66–71, 1998.

[12] H. Ramalhinho Lourenço. Job-shop scheduling: Computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83:347–364, 1995.

[13] H. Ramalhinho Lourenço and M. Zwijnenburg. Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory & Applications*, pages 219–236. Kluwer Academic Publishers, 1996.

[14] O. Martin and S. W. Otto. Partitoning of unstructured meshes for load balancing. *Concurrency: Practice and Experience*, 7:303–314, 1995.

[15] O. Martin and S. W. Otto. Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63:57–75, 1996.

[16] O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, 1991.

[17] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, 1997.

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

[19] E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91:160–175, 1996.

[20] F. Ogbu and D. Smith. The application of the simulated annealing algorithm to the solution of the $n/m/C_{\max}$ flowshop problem. *Computers & Operations Research*, 17(3):243–253, 1990.

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

[22] C. R. Reeves. Improving the efficiency of tabu search for machine sequencing problems. *Journal of the Operational Research Society*, 44(4):375–382, 1993.

[23] C. R. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1):5–13, 1995.

[24] C. R. Reeves and T. Yamada. Genetic algoriothms, path relinking, and the flowshop sequencing problem. *Evolutionary Computation*, 6(1):45–60, 1998.

[25] T. Stützle. An ant approach to the flow shop problem. In *Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98)*, volume 3, pages 1560–1564. Verlag Mainz, Wissenschaftsverlag, Aachen, 1997.

[26] É. D. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47:65–74, 1990.

[27] É. D. Taillard. Benchmarks for Basic Scheduling Problems. *European Journal of Operational Research*, 64:278–285, 1993.

[28] S. Turner and D. Booth. Comparison of heuristics for flow shop sequencing. *OMEGA*, 15(1):75–85, 1987.

[29] F. Werner. On the heuristic solution of the permutation flow shop problem by path algorithms. *Computers & Operations Research*, 20(7):707–722, 1993.

[30] M. Widmer and A. Hertz. A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*, 41:186–193, 1989.

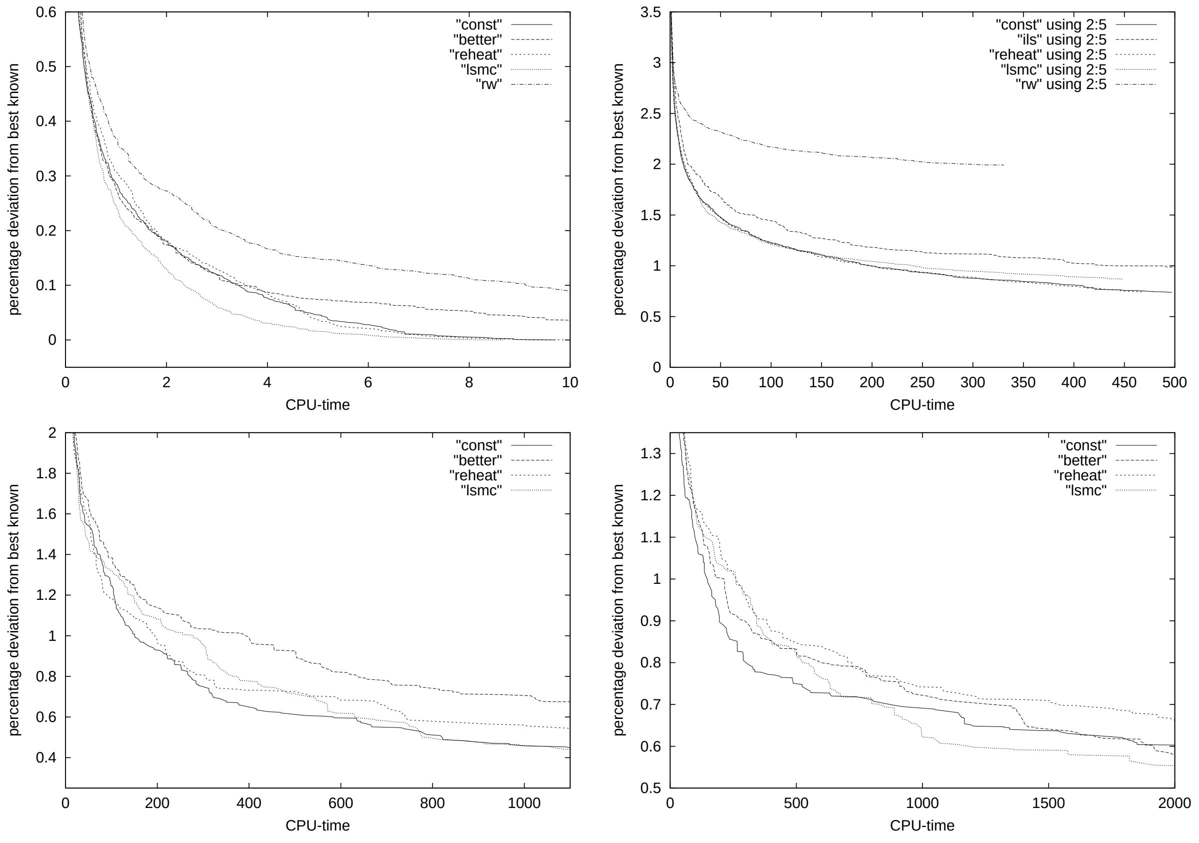

Figure 2: Comparison of acceptance criteria for the ILS algorithm. Given is the trade-off between CPU-time (given on the x-axis) and solution quality (given on the y-axis as the percentage deviation from the best-known solution or the optimum in the case of `ta021`). Plots are for `ta021` (upper left), `ta053` (upper right), `ta087` (lower left), and `ta105` (lower right).

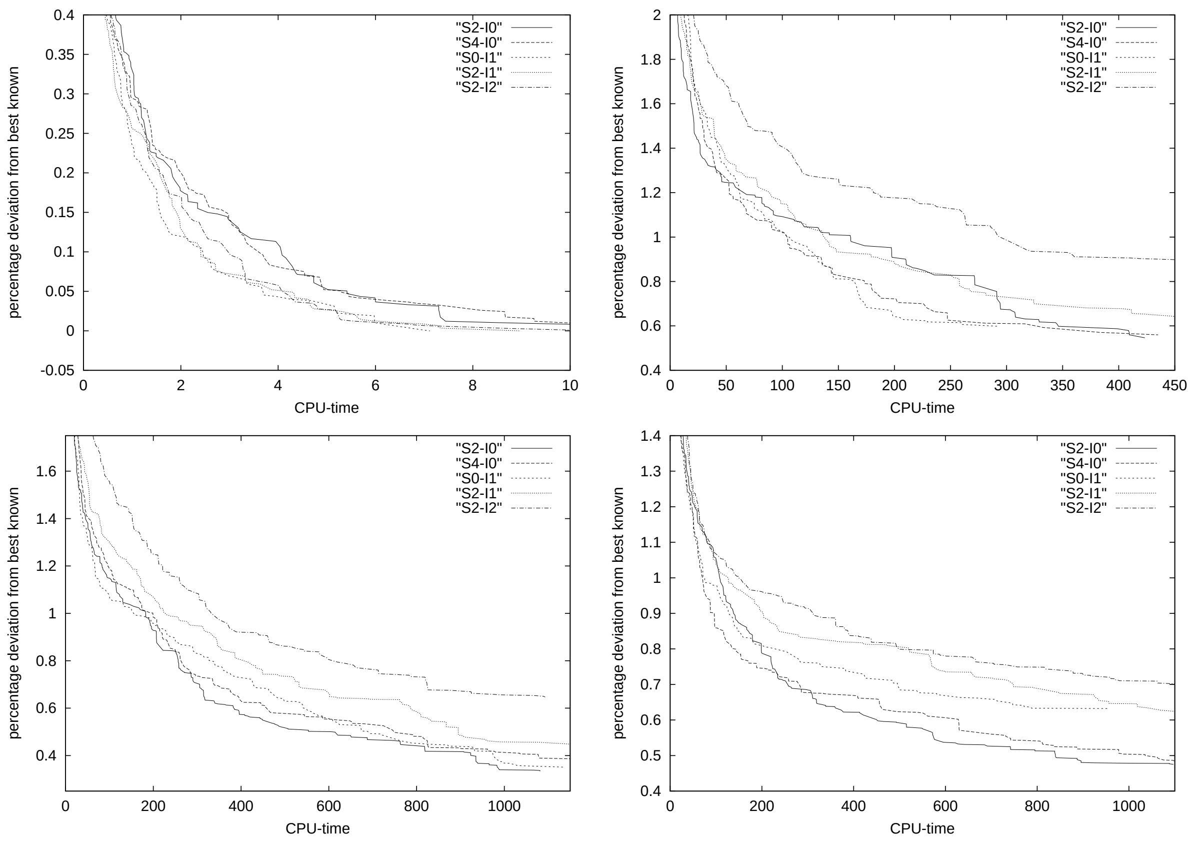

Figure 3: Investigation of the appropriate kick-move strength for the ILS algorithm using acceptance criterion *ConstTemp*(s, s''). Given is the trade-off between CPU-time (given on the x-axis) and solution quality (given on the y-axis as the percentage deviation from the best-known solution or the optimum in the case of `ta021`). Plots are for `ta021` (upper left), `ta053` (upper right), `ta087` (lower left), and `ta105` (lower right). $Sx - Iy$ indicates that one kick-move is composed of x swap-moves and y interchange-moves.