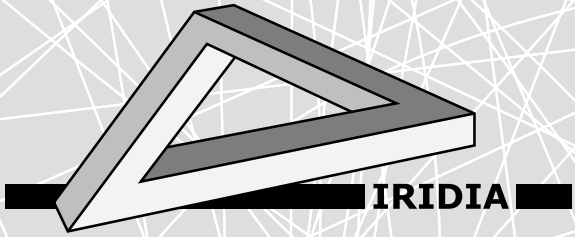

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

An Iterated Greedy Heuristic for the Sequence Dependent Setup Times Flowshop Problem with Makespan and Weighted Tardiness Objectives

Rubén Ruiz and Thomas Stützle

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2006-002

January 2006

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:
IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2006-002

An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives

Rubén Ruiz[1*], Thomas Stützle[2]

[1] Universidad Politécnica de Valencia. Departamento de Estadística e Investigación Operativa Aplicadas y Calidad. Grupo de Investigación Operativa.
Camino de Vera S/N, 46021 Valencia (Spain)
email: rruiz@eio.upv.es

[2] Université Libre de Bruxelles. IRIDIA, CP 194/6
Avenue Franklin Roosevelt 50. 1050 Bruxelles (Belgium)
email: stuetzle@iridia.ulb.ac.be

Abstract

Iterated Greedy (IG) algorithms are based on a very simple principle, are easy to implement and can show excellent performance. In this paper, we propose two new IG algorithms for a complex flowshop problem that results from the consideration of sequence dependent setup times on machines, a characteristic that is often found in industrial settings. We propose two IG algorithms; the first is a straightforward adaption of the IG principle, while the second incorporates a simple descent local search. Furthermore, we consider two different optimization objectives, the minimization of the maximum completion time or makespan and the minimization of the total weighted tardiness. Extensive experiments and statistical analyses demonstrate that, despite their simplicity, the IG algorithms are for both objectives new state-of-the-art algorithms.

Keywords: Iterated Greedy, Flowshop Scheduling, Sequence Dependent Setup Times, Makespan, Weighted Tardiness, Stochastic Local Search, Metaheuristics.

1 Introduction

In the flowshop problem we have a set $N = \{1, \ldots, n\}$ of n independent jobs that have to be processed on a set $M = \{M_1, \ldots, M_m\}$ of m machines. Without loss of generality, the machines are arranged in order so that all jobs are processed sequentially, i.e., first on machine 1, then on machine 2 and so on until the last machine m, which gives a total of $n \cdot m$ operations. Each operation requires a known, fixed amount of time that is commonly denoted as p_{ij}, $i \in M$, $j \in N$.

In the flowshop literature, one can find an overwhelming number of papers for the regular flowshop problem with the objective of minimizing the maximum completion time across all jobs (also called makespan or denoted by C_{max}) as well as for most variants like no-wait, limited buffer and so on. However, the sequence dependent flowshop problem (SDST flowshop) has attracted much less attention. Specific to the SDST flowshop are the setup times. Setup times involve

*Corresponding author

operations that have to be performed on machines and that are not part of the processing times. This includes cleaning, fixing or releasing parts to machines, adjustments to machines, etc. The most complex situation is when the setup times are separable from the processing time (i.e., the setup operations can be done before the product arrives to the machine) and when they are sequence dependent (i.e., the amount of setup time depends on the machine, on the job that the machine was processing and on the job that comes next). An example for the SDST flowshop comes from the paint industry; after producing a black paint, substantial cleaning must be performed if one intends to produce white paint, while less cleaning is necessary if a batch of dark grey paint is to be produced. In this paper, we will consider this last type of setup times and denote by S_{ijk} the known, fixed and non-negative setup time on machine i when producing job k, $k \in N$, after having produced job j.

The objective in flowshop scheduling problems is to find a sequence for processing the jobs on the machines so that a given criterion is optimized. This yields a total of $n!$ possible orderings of the operations on each machine and a total of $(n!)^m$ possible processing sequences. Usually, as also done here, only the so-called permutation sequences are considered, where the processing order of operations is the same for all machines. In this paper, we will consider two independent objectives, namely the minimization of the makespan and the minimization of the total weighted tardiness. According to Pinedo (2002), the two problems are denoted as $F/S_{ijk}, prmu/C_{max}$ and $F/S_{ijk}, prmu/\sum_{j=1}^{n} w_j T_j$, respectively; in the following, we will refer to these two problems as SDST-FSP-C_{max} and SDST-FSP-WT, respectively. Makespan is one of the most commonly tackled objectives. In the flowshop context, minimizing the maximum completion time is equal to minimizing the idle time of machines. This objective is also closely related to the maximization of the throughput, which is very important in production environments where machines are expensive to purchase and operate. However, the best sequence with respect to makespan minimization might have a large number of jobs being completed after their due dates. Hence, we also consider the minimization of the weighted tardiness. With this second, independent objective, every job has a weight or priority that indicates its relative importance. This priority is multiplied by the amount of time the job is completed after its due date. Weighted tardiness minimization is of uttermost importance in make-to-order or just-in-time environments, where due dates are typically set by clients.

As regards the computational complexity, the SDST flowshop with the C_{max} objective has been shown to be $\mathcal{NP}$-hard by Gupta (1986) even when $m = 1$ and also when $m = 2$ and setups are present only on the first or second machine (see Gupta and Darrow, 1986). Based on the complexity hierarchies for scheduling problems from Pinedo (2002), we can deduce that also the SDST flowshop with total weighted tardiness objective is $\mathcal{NP}$-hard.

Recently, a rather new heuristic method, called Iterated Greedy (IG), has shown state-of-the-art performance for the permutation flowshop scheduling problem where the goal is to minimize the makespan (FSP-C_{max}) Ruiz and Stützle (2006). This is rather noteworthy since a large number of often rather complex and very fine-tuned algorithms have been proposed for the FSP-C_{max} (Ruiz and Maroto, 2005) and the IG method is remarkably simple. IG for the FSP-C_{max} works by iteratively applying two phases; in a *destruction* phase, some jobs are eliminated from the current solution, and in the *construction* phase the eliminated jobs are reinserted into the sequence using a greedy construction heuristic. To these two phases, an additional local search phase can be added in a straightforward way.

The main goal of this article is an experimental study as to whether the high performance of IG for the PFSP is transferable also to other, more complex flowshop environments, in particular SDST flowshop problems. In fact, the results obtained with the proposed IG algorithms show that the excellent performance on the PFSP also is transferable to these, more complex scheduling

environments and our proposed IG algorithms can be identified as new state-of-the-art algorithms for the SDST-FSP-C_{max} and the SDST-FSP-WT.

The remainder of the paper is organized as follows: Section 2 reviews the literature on the two problems considered here. In Section 3, we introduce the two IG algorithms proposed in this paper. A full experimental evaluation and comparison of the proposed methods is shown, along with statistical analyses, in Section 4. Finally, Section 5 closes this study with some conclusions and future research directions.

2 Literature review

Compared to the regular flowshop, on which hundreds of papers have been published, the literature on the SDST counterpart is scarce. In a recent article, Ruiz et al. (2005) carried out an extensive literature survey about this problem and here we summarize the most important aspects.

Exact techniques for the SDST flowshop have shown rather limited results. The works of Corwin and Esogbue (1974) on dynamic programming or Srikar and Ghosh (1986), Stafford and Tseng (1990) and Tseng and Stafford (2001) on integer programming or the papers of Ríos-Mercado and Bard (1998a) and Ríos-Mercado and Bard (1999a) on branch and bound methods allow to solve optimally problem instances with only about 10 jobs in general and in most cases only the C_{max} objective is considered. Some heuristics and applications of stochastic local search algorithms to the SDST-FSP-C_{max} have also been proposed. Simons (1992) proposed two general heuristics for the SDST flowshop, named TOTAL and SETUP. Das et al. (1995) proposed a heuristic method based on a savings index. Ríos-Mercado and Bard (1998b) proposed a modification of the well known NEH heuristic for the regular flowshop from Nawaz et al. (1983) that considered setup times and that they called NEH_RMB; in the same article they proposed a GRASP algorithm. In a later work, the same authors proposed a modification of the heuristics of Simons (1992) resulting in a new heuristic called HYBRID (Ríos-Mercado and Bard, 1999b). Recently, Ruiz et al. (2005) proposed a genetic and a memetic algorithm for the SDST-FSP-C_{max}. They carried out an extensive experimental study and compared these two algorithms to all aforementioned algorithms as well as to many adaptations of other methods that were proposed for the FSP-C_{max}. The result was that both algorithms, especially the memetic, are clearly superior to all other alternatives.

On the SDST-FSP-WT, little has been published. In two similar articles (Parthasarathy and Rajendran, 1997a, Parthasarathy and Rajendran, 1997b), a Simulated Annealing heuristic was proposed for the SDST flowshop problem with the objectives of minimizing the maximum weighted tardiness and the total weighted tardiness. In a more recent work, Rajendran and Ziegler (2003) introduced a new heuristic paired with a local search improvement scheme for a combined objective of total weighted flow-time and tardiness. Another similar work is Rajendran and Ziegler (1997) were only weighted flow-time is considered.

Although the existing methods for the SDST-FSP-C_{max} have been comparatively studied in Ruiz et al. (2005), no similar comparison has been done with other objectives.

3 IG for the SDST Flowshop

Iterated Greedy algorithms start from some initial solution and then iterate through a main loop in which first a partial candidate solution s_p is obtained by removing a number of solution components from a complete candidate solution s and next a complete solution s' is re-constructed

```
procedure Iterated_Greedy
    s_0 := GenerateInitialSolution;
    s := LocalSearch(s_0);          %optional
    repeat
        s_p := Destruction(s)
        s' := Construction(s_p)
        s' := LocalSearch(s')       %optional
        s := AcceptanceCriterion(s, s')
    until termination condition met
end
```

Figure 1: General outline of an Iterated Greedy algorithm.

starting with s_p. Before continuing with the next loop, an acceptance criterion decides whether s' becomes the new incumbent solution. This process is repeated until some stopping criterion like a maximum number of iterations or a computation time limit is met. It is also straightforward to add an optional local search phase for improving the re-constructed solution s' before applying the acceptance test. An outline of a generic IG algorithm is given in Figure 1.

IG is related to other Stochastic Local Search (SLS) methods (see Hoos and Stützle, 2004), and especially to Iterated Local Search (ILS) (see Lourenço et al., 2003). The applications of the IG method are, for the moment rather limited. Jacobs and Brusco (1995) and Marchiori and Steenbeek (2000) applied IG algorithms to Set Covering problems and recently, Ruiz and Stützle (2006) applied a simple IG method with and without the optional local search phase to the FSP-C_{max}. In this latter work, the simple IG algorithm yielded results far better than several other, much more complex approaches. Therefore, in the following we provide the description of a new IG algorithm version that we apply to the more realistic SDST flowshop variant under the two already mentioned objectives.

Before entering into more detail, we give additional notation and describe how to calculate the C_{max} and $\sum_{j=1}^{n} w_j T_j$ objectives of a given sequence. For every job j, $j \in N$ we have a due date denoted as d_j and a weight w_j. We denote with $C_{i,j}$ the completion time of job j on machine i. The completion time of job j at the last stage m is $C_{m,j}$ or C_j in short. The *lateness* of job j is defined as $L_j = C_j - d_j$. Note that L_j can be positive or negative and, hence, the *tardiness* of a job j is defined as $T_j = \max\{L_j, 0\}$.

Let π be a permutation or sequence of the n jobs. The job in position j of the sequence is denoted as $\pi(j)$. The completion times for the jobs on the machines are calculated with the formula:

$$C_{i,\pi(j)} = \max\{C_{i,\pi(j-1)} + S_{i,\pi(j-1),\pi_{(j)}}; C_{i-1,\pi(j)}\} + p_{i,\pi(j)} \quad (1)$$

where, $C_{0,\pi(j)} = 0$, $S_{i,0,\pi(j)} = 0$ and $C_{i,0} = 0$, for all $i \in M, j \in N$. With this we have that $C_{max} = C_{\pi(n)}$ and with the completion times on the last machine for all jobs we can calculate the total weighted tardiness objective as $\sum_{j=1}^{n} w_j T_j$.

3.1 Algorithm initialization

The initial solution for IG is ideally generated by a high performance construction heuristic. For the SDST-FSP-C_{max}, we used the NEH_RMB heuristic proposed by Ríos-Mercado and Bard

(1998b); it is an extension of the well-known NEH heuristic from Nawaz et al. (1983), which still can be considered as the best construction heuristic for the FSP-C_{max}, as shown in Ruiz and Maroto (2005). It is noteworthy that NEH_RMB has a computational complexity of $\mathcal{O}(n^2m)$, since for the SDST-FSP-C_{max} the same speed-ups as proposed by Taillard (1990) for the basic NEH can be applied. Recall that the NEH is an insertion heuristic, where at each step the next unscheduled job is inserted in each possible position of some partial solution. The job is then inserted in the position where the scheduling criterion takes the lowest value. For executing such an insertion heuristic, first the jobs have to be ordered in some way.

For the SDST-FSP-WT, the situation is rather different, since there are no known high performing construction heuristics available. Hence, we extended the original NEH, which has been widely used for many other scheduling problems, to this second problem, resulting in a heuristic we call NEH_EWDD. However, the accelerations proposed by Taillard (1990) are not transferable for objectives other than C_{max} and, hence, the complexity of NEH_EWDD is $\mathcal{O}(n^3m)$, the same as that of the original NEH. In NEH_EWDD we consider the existence of weights and due dates for defining an initial order in which the jobs are considered for insertion. The initial order in NEH_EWDD is based on the Earliest Weighted Due Date dispatching rule (EWDD) that arranges jobs in ascending order of their weighted due dates, i.e., d_j/w_j.

3.2 Destruction, construction and acceptance criterion

As said above, once the initial sequence has been obtained, the destruction step is applied. Let the initial sequence be π. In this step, a given number d of jobs, chosen at random and without repetition, are removed from the sequence. This creates two subsequences, the first one of size d is called π_R and contains the removed jobs in the order in which they were removed. The second, of size $n-d$, is the original one without the removed jobs, that we call π_D.

After the two subsequences are created, the construction step is applied. Here, all jobs in π_R are reinserted in π_D by applying the NEH heuristic, i.e., the first job of π_R ($\pi_{R(1)}$) is inserted in all possible $n-d+1$ positions of π_D generating $n-d+1$ partial sequences that include job $\pi_{R(1)}$. The sequence with the best objective value is kept for the following iteration. The process ends when π_R is empty and therefore π_D is again of size n. It has to be noted that for the SDST-FSP-C_{max} we use in this construction phase the speedier NEH_RMB heuristic.

Another step in the Iterated Greedy algorithm is to decide whether the reconstructed sequence is accepted or not as the incumbent solution for the next iteration. A pure descent criterion would be to accept solutions with better objective function values. However, this acceptance criterion is prone to stagnation. As an alternative we can consider a simple simulated annealing-like acceptance criterion with constant temperature. This has been used in other works like in Osman and Potts (1989), Stützle (1998) or Ruiz and Stützle (2006). The constant Temperature depends on the processing times, the number of jobs and machines, and an adjustable parameter T:

$$\mathsf{Temperature} = T \cdot \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}}{n \cdot m \cdot 10}. \tag{2}$$

We denote by $O(incumbent)$ and $O(\pi)$ the respective objective function values of the incumbent sequence and the reconstructed sequence. If $O(\pi) \geq O(incumbent)$, π is accepted as the new incumbent if:

$$random \leq \exp\{O(\pi) - O(incumbent)/\mathsf{Temperature}\}$$

where $random$ is a random number uniformly distributed in $[0, 1]$.

```
procedure LocalSearch_Insertion (π)
    improve := true;
    while (improve = true) do
        improve := false;
        for i := 1 to n do
            remove a job k at random from π (without repetition)
            π' := best permutation obtained by inserting k in any possible positions of π;
            if O(π') < O(π) then
                π := π';
                improve := true;
            endif
        endfor
    endwhile
    return π
end
```

Figure 2: Descent local search procedure based on the insertion neighborhood

3.3 Hybridization with local search

To further improve the performance of the basic IG algorithm, Ruiz and Stützle (2006) added a simple descent local search phase after after the NEH initialization and after each sequence reconstruction. In fact, such a hybridization of construction heuristics with local search is common practice in stochastic local search. We apply a local search based on the insertion neighborhood. This neighborhood has been regarded as superior to the swap or exchange neighborhoods in flowshop scheduling by many authors (see Osman and Potts, 1989, Taillard, 1990, Stützle, 1998 or Ruiz et al. (2005) among many others). The insertion neighborhood of a sequence comprises all those sequences that can be obtained by removing some job and inserting it in another position. We implemented a local search algorithm that searches this neighborhood in a particular way: In each local search step, a job is removed from the sequence (at random and without repetition) and then inserted in all possible n positions. The current sequence π is replaced by the best sequence among the n possible ones, only if an improvement of π can be obtained. The procedure ends when no further improvements are found, i.e., when the sequence is a local optimum with respect to the insertion neighborhood. The pseudocode of this procedure is shown in Figure 2.

With this optional local search step, the proposed IG algorithm is depicted in Figure 3.

4 Experimental evaluation

For the proposed, basic IG algorithm, which we refer to as IG_RS, we set the temperature parameter T to 0.5 and the number of jobs that are extracted from the sequence at each iteration, (d) to 4. This follows the extensive experimental study carried out in Ruiz and Stützle (2006). In some short trial runs, we tested other values for these two parameters with no apparent improvement. Therefore, we keep the original values as stated.

If we use the optional local search, we refer to the resulting algorithm as IG_RS$_{LS}$. In the following, we present the results of both algorithms compared to other existing methods for each of the two objectives separately.

```
procedure IteratedGreedy_for_SDST_flowshop
   π := NEH_RMB or NEH_EWDD;        % Initialization
   π := LocalSearch_Insertion(π);
   π_b := π;
   while (termination criterion not satisfied) do
      π' := π;                      % Destruction phase
      for i := 1 to d do
         π' := remove one job at random from π' and insert it in π'_R;
      endfor
      for i := 1 to d do            % Construction phase
         π' := best permutation obtained by inserting job π_{R_(i)} in all possible positions of π';
      endfor
      π'' := LocalSearch_Insertion(π');   % Local Search
      if O(π'') < O(π) then         % Acceptance Criterion
         π := π'';
         if O(π) < O(π_b) then      % check if new best permutation
            π_b := π;
         endif
      elseif (random ≤ exp{−(O(π'') − O(π))/Temperature}) then
         π := π'';
      endif
   endwhile
   return π_b
end
```

Figure 3: Iterated Greedy algorithm with the optional local search phase.

4.1 Experimental results for the SDST-FSP-C_{max}

Ruiz et al. (2005) generated four instance sets for testing algorithms for the SDST-FSP-C_{max}. Each set is based on the original 120 instances of Taillard (1993), which are organized in 12 groups with 10 instances each. The groups contain different combinations of the number of jobs n and the number of machines m. The combinations are: $\{20, 50, 100\} \times \{5, 10, 20\}$, $200 \times \{10, 20\}$ and 500×20. The processing times (p_{ij}) in Taillard's instances are generated from a uniform distribution in the range $[1, 99]$. To generate the four instance sets for the SDST-FSP-C_{max}, the ratios of the setup times to the processing times were changed such that the sequence dependent setup times are at most 10%, 50%, 100% or 125%, respectively, of the maximum possible processing times of Taillard's original instances. This results in the four sets called SDST10, SDST50, SDST100, and SDST125 that have the setup times uniformly distributed in the range $[1, 9]$, $[1, 49]$, $[1, 99]$ and $[1, 124]$, respectively. Thus, we have a total of 480 different instances, which can be downloaded from `http://www.upv.es/gio/rruiz`.

We are going to test a total of six algorithms. The first two are the proposed IG_RS and IG_RS$_{LS}$ algorithms. We will also test the genetic and memetic algorithms of Ruiz et al. (2005). We refer to the genetic algorithm as GA. The memetic algorithm in the original work used a curtailed local search to supposedly improve the speed of the algorithm. In this article, we propose a modified version that uses the same local search as IG_RS$_{LS}$ in the local search

phase of the memetic algorithm. We refer to the original memetic algorithm as MA and to the modified memetic algorithm as MA_{LS}. The last algorithm that we test is the PACO ant colony optimization algorithm from Rajendran and Ziegler (2004). PACO was adapted to the SDST-FSP-C_{max} and the initialization was changed from the original NEH to NEH_RMB. It has to be noted that we could have tested also other SLS algorithms that were proposed for the SDST flowshop (see Section 2 for details). However, from the extensive comparison in Ruiz et al. (2005), which tested a total of 14 methods, including many SDST flowshop specific methods as well as regular flowshop adapted algorithms, the memetic algorithm (denoted here as MA) resulted to be clearly superior to all others. Consequently, here we will mainly compare to that algorithm.

For the experiments we use a PC/AT computer with an Athlon XP 1600+ processor (1400 MHz) and 512 MBytes of main memory. Every algorithm is run 10 independent times with a stopping criterion based on an elapsed CPU time given by the formula $(n \cdot m/2) \cdot t$ milliseconds. This allows for more time as the number of jobs n and the number of machines m grows. We will carry out three different experiments setting t to 30, 60 and 90. So, for example, for the largest instances (500×20) and for $t = 90$ we will have that each of the 10 repetitions of each algorithm takes $(500 \cdot 20/2) \cdot 90 = 450,000$ milliseconds or 7.5 minutes. The response variable for each repetition and each of the 480 instances is the relative percentage deviation over the best known solution and it is calculated as follows:

$$\text{Relative Percentage Deviation } (RPD) = \frac{Some_{sol} - Best_{sol}}{Best_{sol}} \cdot 100, \tag{3}$$

where $Some_{sol}$ is the solution returned for a given instance and $Best_{sol}$ is the best known solution for the same instance. The best known solutions for the four instance sets can be downloaded from `http://www.upv.es/gio/rruiz`. The results for instance sets SDST10 and SDST50 are shown in Table 1 and for instance sets SDST100 and SDST125 in Table 2. For each combination of n and m, these results are averaged across the 10 instances and across all repetitions. Therefore, in each cell of the tables we show the results for each value of t across $10 \cdot 10 = 100$ different values.

As it can be seen, for all algorithms the average relative percentage deviations ($AVRPD$) increase as the ratio of setup times to processing time increases; for example, in the SDST10 set the highest $AVRPD$ is 1.39 for the GA and $t = 30$, whereas in SDST125 the highest $AVRPD$ is 3.97. The same can be said about the lowest $AVRPD$. Additionally, we see that as t increases, the benefits in SDST10 are not as marked as are in SDST100 or SDST125. An interpretation of this fact is that the problems may become increasingly difficult as the ratio of setup times to processing time increases.

From the cross averages in the tables (last line for each instance set), we can see that GA is overall the worst performer, followed by IG_RS, which is second worst. Both algorithms do not incorporate local search but IG_RS is much simpler and easier to implement than the GA– hence, IG_RS is preferable to GA and, hence, the best performing algorithm that does not use an explicit local search.

Another salient conclusion from the experiments comes after the comparison of the algorithms MA_{LS} and MA: the MA_{LS} memetic algorithm with the more intensive local search performs much better than MA as it obtains better results by a considerably margin. This observation can be substantiated by statistical tests, as we show below. As a matter of fact, considering all 120 instances for each of the four groups, the three values of t and the 10 repetitions carried out (14400 values), MA_{LS} improved the results of MA in 7,591 occasions, while both algorithms yielded the same solution in 1,394 cases and only in 5,415 cases MA was better, and even in these latter cases the differences between MA and MA_{LS} were rather minor.

PACO has a peculiar profile. While in SDST10 it yields on average better solutions than GA,

Instance	GA	MA	MA_{LS}	PACO	IG_RS	IG_RS_{LS}
SDST10 Instances						
20×5	0.43/0.46/0.41	0.49/0.90/0.70	0.12/0.10/0.08	0.18/0.21/0.18	0.21/0.19/0.14	0.08/0.05/0.04
20×10	0.59/0.57/0.56	0.55/0.28/0.36	0.13/0.13/0.13	0.33/0.26/0.22	0.28/0.22/0.24	0.08/0.05/0.04
20×20	0.44/0.37/0.39	0.59/0.52/0.56	0.14/0.09/0.10	0.20/0.16/0.12	0.30/0.22/0.19	0.07/0.05/0.04
50×5	1.04/0.93/0.92	0.77/0.57/0.77	0.43/0.31/0.30	0.53/0.44/0.42	1.00/0.88/0.84	0.37/0.32/0.27
50×10	2.10/2.07/2.01	1.21/1.38/1.26	1.12/0.83/0.81	1.23/1.02/1.06	1.58/1.58/1.43	0.76/0.60/0.53
50×20	2.23/2.18/2.10	1.38/1.21/1.28	1.16/0.96/0.82	1.27/1.06/1.01	1.85/1.70/1.54	0.91/0.64/0.60
100×5	1.28/1.10/1.03	0.76/0.70/0.63	0.54/0.40/0.31	0.87/0.80/0.76	1.44/1.36/1.34	0.43/0.38/0.33
100×10	1.48/1.39/1.33	0.91/0.81/0.90	0.78/0.60/0.48	0.99/0.84/0.77	1.49/1.37/1.32	0.61/0.44/0.38
100×20	2.07/1.93/1.83	1.49/1.11/1.06	1.27/0.97/0.82	1.49/1.25/1.12	1.75/1.48/1.47	0.88/0.71/0.54
200×10	1.63/1.42/1.32	0.81/0.73/0.65	0.79/0.61/0.48	1.04/0.94/0.85	1.50/1.39/1.33	0.58/0.43/0.32
200×20	2.00/1.79/1.71	1.14/0.93/0.87	1.11/0.87/0.76	1.31/1.10/0.95	1.45/1.25/1.12	0.79/0.53/0.38
500×20	1.38/1.31/1.27	0.74/0.54/0.48	0.69/0.54/0.43	0.82/0.69/0.61	1.01/0.88/0.82	0.46/0.31/0.21
Average	1.39/1.29/1.24	0.90/0.81/0.79	0.69/0.53/0.46	0.86/0.73/0.67	1.16/1.04/0.98	0.50/0.38/0.31
SDST50 Instances						
20×5	1.34/1.30/1.15	0.44/1.21/1.50	0.37/0.35/0.30	0.58/0.53/0.51	0.83/0.69/0.58	0.26/0.18/0.10
20×10	1.21/1.16/1.17	0.92/0.87/0.77	0.41/0.31/0.32	0.49/0.43/0.44	0.66/0.62/0.58	0.28/0.20/0.19
20×20	0.57/0.57/0.49	0.87/0.23/0.78	0.20/0.16/0.16	0.35/0.32/0.25	0.60/0.41/0.37	0.10/0.09/0.07
50×5	3.85/3.57/3.43	2.27/1.65/2.18	1.79/1.39/1.13	2.52/2.05/1.98	2.99/2.61/2.42	1.41/1.13/1.04
50×10	3.24/3.15/3.01	1.81/1.96/1.68	1.49/1.24/1.08	2.15/1.81/1.62	2.44/2.23/2.12	1.33/1.17/0.92
50×20	2.57/2.49/2.43	1.93/1.61/1.69	1.33/1.07/0.89	1.65/1.42/1.28	2.34/2.06/2.03	1.16/0.93/0.82
100×5	4.64/4.06/3.98	2.64/2.35/2.34	2.23/1.72/1.38	4.37/4.11/3.95	2.93/2.67/2.33	1.51/1.27/1.09
100×10	3.61/3.24/3.07	2.20/1.82/1.52	1.84/1.53/1.21	3.44/3.19/3.10	2.69/2.23/2.13	1.37/1.04/0.88
100×20	2.96/2.71/2.51	2.00/1.66/1.54	1.73/1.35/1.03	2.87/2.66/2.45	2.38/2.01/1.82	1.29/0.96/0.81
200×10	3.95/3.64/3.49	1.98/1.71/1.35	1.88/1.43/1.21	3.62/3.48/3.37	2.59/2.19/1.90	1.33/0.88/0.63
200×20	3.04/2.82/2.67	1.62/1.34/1.19	1.61/1.17/1.02	2.89/2.78/2.64	2.07/1.77/1.51	1.10/0.74/0.53
500×20	2.14/2.09/2.07	1.29/0.99/0.76	1.23/0.96/0.79	2.00/2.00/2.00	1.79/1.47/1.28	0.86/0.50/0.31
Average	2.76/2.57/2.46	1.66/1.45/1.44	1.34/1.06/0.88	2.24/2.06/1.97	2.03/1.75/1.59	1.00/0.76/0.62

Table 1: Average relative percentage deviation from the best known solutions for the algorithms for the SDST-FSP-C_{max} on instance sets SDST10 and SDST50 with the termination criteria set at $(n \cdot m/2) \cdot t$ milliseconds maximum CPU time where $t = 30, 60$ and 90, respectively.

IG_RS and MA, its performance deteriorates much more rapidly than all other algorithms as the ratio of setup times to processing times increases. In SDST125, the performance of PACO is worse than IG_RS which is much simpler and does not incorporate any kind of local search as PACO does. This behavior may be due to the type of information used when constructing solutions in PACO. Essentially, PACO exploits the information at which position a job is in a good sequence, while it does not consider the successor / predecessor relationship between jobs. However, as the relative influence of the setup times increases, this latter information becomes probably more important, which explains also the relatively poor performance of PACO.

Finally, we can see from the tables that IG_RS_{LS} gives results that are far better than all other algorithms, including the improved MA_{LS}. For $t = 90$ we have that the overall *AVRPD* for SDST10 and for the algorithms MA, MA_{LS} and IG_RS_{LS} are 0.79, 0.46 and 0.31 respectively, which means that IG_RS_{LS} gives results which are 155% better than MA, which was shown in Ruiz et al. (2005) to be far better than the previous state-of-the-art at that time. Also, IG_RS_{LS} gives results that are 48% better than the much improved MA_{LS}. Hence, the proposed IG_RS_{LS} algorithm can be regarded as the best performer for the considered problem and, at the same time, it is, except for IG_RS, the simplest and easiest to code of those compared.

While a considerable amount of testing has been carried out, comparing algorithms on the

Instance	GA	MA	MA_{LS}	PACO	IG_RS	IG_RS_{LS}
SDST100 Instances						
20×5	2.01/1.88/1.82	1.29/1.73/1.43	0.43/0.37/0.39	0.82/0.71/0.61	1.53/1.48/1.24	0.30/0.25/0.17
20×10	1.48/1.26/1.27	0.87/0.88/1.09	0.31/0.28/0.29	0.66/0.47/0.48	1.42/1.01/1.03	0.35/0.25/0.18
20×20	1.08/1.00/0.94	0.62/0.28/1.14	0.29/0.26/0.17	0.52/0.41/0.48	0.95/0.92/0.74	0.27/0.18/0.17
50×5	5.00/5.35/5.26	3.12/2.65/3.02	2.37/2.24/1.99	3.79/3.40/3.31	3.83/3.89/3.70	1.95/1.95/1.82
50×10	4.19/4.21/4.18	2.59/2.72/2.55	1.98/1.66/1.50	3.05/2.73/2.49	3.10/3.09/2.99	1.57/1.48/1.30
50×20	3.39/3.23/3.11	1.78/2.11/1.77	1.66/1.35/1.18	2.51/2.14/1.98	2.76/2.58/2.40	1.41/1.28/1.11
100×5	6.49/5.99/6.00	3.63/3.50/3.04	3.20/2.69/2.16	6.86/6.89/6.65	3.93/3.82/3.48	2.16/1.95/1.63
100×10	4.58/4.39/4.15	3.03/2.67/2.45	2.26/2.01/1.61	5.14/4.96/4.89	3.28/2.95/2.77	1.61/1.44/1.02
100×20	3.73/3.67/3.49	2.37/2.31/2.39	2.12/2.03/1.53	4.04/4.04/3.91	2.76/2.65/2.46	1.41/1.35/1.05
200×10	5.12/4.95/4.71	2.56/2.31/2.19	2.53/2.19/1.77	5.48/5.62/5.53	2.98/2.86/2.49	1.67/1.25/0.92
200×20	3.59/3.65/3.48	1.99/1.81/1.68	1.93/1.68/1.40	3.70/3.87/3.82	2.27/2.08/1.92	1.26/0.93/0.76
500×20	2.50/2.66/2.64	1.53/1.44/1.16	1.53/1.35/1.14	2.50/2.75/2.75	1.87/1.70/1.50	0.96/0.73/0.46
Average	3.60/3.52/3.42	2.11/2.03/1.99	1.72/1.51/1.26	3.26/3.17/3.07	2.56/2.42/2.23	1.24/1.09/0.88
SDST125 Instances						
20×5	2.06/1.80/1.90	1.69/2.05/1.40	0.67/0.34/0.32	0.88/0.64/0.65	1.96/1.40/1.24	0.46/0.35/0.30
20×10	1.74/1.66/1.52	1.02/1.48/1.24	0.51/0.42/0.37	0.85/0.68/0.56	1.62/1.39/1.44	0.53/0.41/0.36
20×20	1.06/0.97/0.95	1.37/0.96/1.21	0.28/0.22/0.24	0.47/0.39/0.39	0.94/0.84/0.81	0.26/0.22/0.19
50×5	6.09/5.83/5.63	3.71/3.97/3.48	2.97/2.47/1.97	4.59/4.07/3.67	4.57/4.25/4.00	2.37/2.18/2.01
50×10	4.64/4.73/4.59	3.14/2.13/3.35	2.07/1.78/1.50	3.60/3.16/2.96	3.95/3.60/3.47	1.94/1.67/1.54
50×20	3.32/3.41/3.25	2.16/2.50/1.63	1.59/1.43/1.26	2.55/2.43/2.06	2.77/2.71/2.59	1.42/1.45/1.18
100×5	7.33/6.86/6.82	4.38/4.45/3.65	3.55/3.02/2.52	8.19/7.89/7.75	4.70/4.58/4.14	2.41/2.27/1.91
100×10	5.33/5.14/4.80	3.24/3.10/2.84	2.78/2.37/1.94	6.02/5.89/5.61	3.66/3.43/3.26	2.07/1.65/1.34
100×20	3.99/3.79/3.50	2.56/2.40/2.16	2.31/1.80/1.50	4.37/4.32/4.15	2.91/2.69/2.60	1.52/1.22/1.00
200×10	5.53/5.65/5.37	2.81/2.76/2.63	2.73/2.51/2.14	5.80/6.27/6.20	3.33/3.17/2.94	1.79/1.60/1.17
200×20	3.86/3.88/3.69	2.08/1.94/1.69	2.04/1.74/1.49	3.93/4.20/4.16	2.51/2.40/2.24	1.38/1.06/0.76
500×20	2.71/2.89/2.83	1.71/1.66/1.36	1.70/1.53/1.23	2.77/3.03/3.02	2.13/1.91/1.64	1.08/0.83/0.52
Average	3.97/3.88/3.74	2.49/2.45/2.22	1.93/1.64/1.37	3.67/3.58/3.43	2.92/2.70/2.53	1.44/1.24/1.02

Table 2: Average relative percentage deviation from the best known solutions for the algorithms for the SDST-FSP-C_{max} on instance sets SDST100 and SDST125 with the termination criteria set at $(n \cdot m/2) \cdot t$ milliseconds maximum CPU time where $t = 30, 60$ and 90, respectively.

basis of means is rather weak. Hence, we have carried out an analysis of experiments using the ANOVA technique. In fact, we did four analyses, one for each instance set (SDST10, ..., SDST125), where the controlled factors are the type of instance, the algorithm applied and t. For each such experiment we consider all algorithms, values of t, instance types (we have 12 types, one for each combination of n and m, i.e., 1: 20×5,...,12: 500×20) and repetitions. The response variable is the relative percentage deviation or RPD. For each experiment all three hypotheses of the ANOVA technique (normality, homocedasticity and independence of the residuals) where checked and accepted. In these experiments, of particular interest is the interaction between the algorithm and t factors. The means plot along with the Least Significant Difference (LSD) confidence intervals (at the 95% confidence level) are given in Figure 4 for experiment SDST10.

As we can see, for all algorithms, increasing the value of t results in better performance although for $t = 60$ and $t = 90$ the differences are very small and even not statistically significant for MA. We can confirm also the previous observation that GA is statistically worse than all other algorithms (recall that overlapping LSD intervals means no statistically significant differences at the 95% confidence). We also observe that MA_{LS} is statistically better than MA. Furthermore, MA_{LS} with $t = 30$ is better than MA with $t = 90$ which is a rather impressing result. From the plot it is also clear than IG_RS is better than GA and also in this case we have that IG_RS

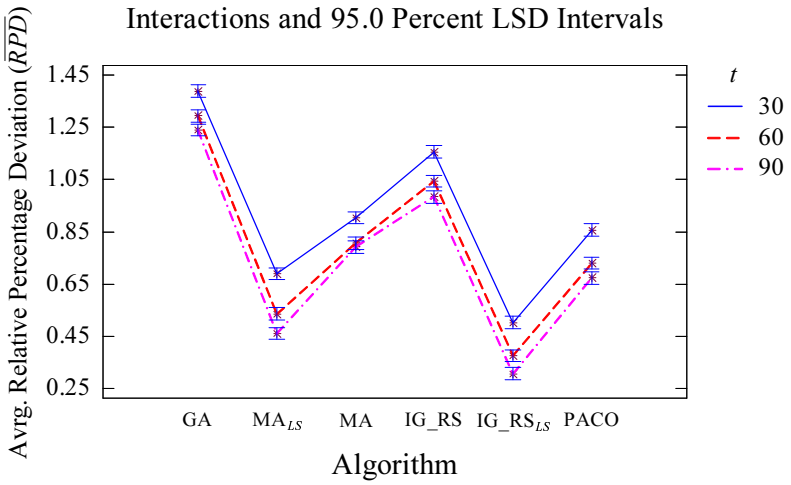

Figure 4: Plot of the average percentage deviation from best known solutions for the interaction between the type of algorithm and the different values of t for the stopping criterion. Instance set SDST10 and makespan criterion.

with $t = 30$ yields better results than GA with $t = 90$. PACO in this experiment is comparable or marginally better than MA but the improved version MA_{LS} yields better results than PACO. Lastly, from the plot it is clear that IG_RS_{LS} is by far the highest performing algorithm and it beats MA_{LS} by a considerable margin. We can even see than IG_RS_{LS} with $t = 60$ gives statistically better results than MA_{LS} with $t = 90$.

In the previous analysis, we observed the results averaged across all instance types, but it is also possible to do a more in-depth analysis. In Figure 5 we see for instance set SDST10 the interaction between the type of instance and the algorithm factors after having fixed t to 90.

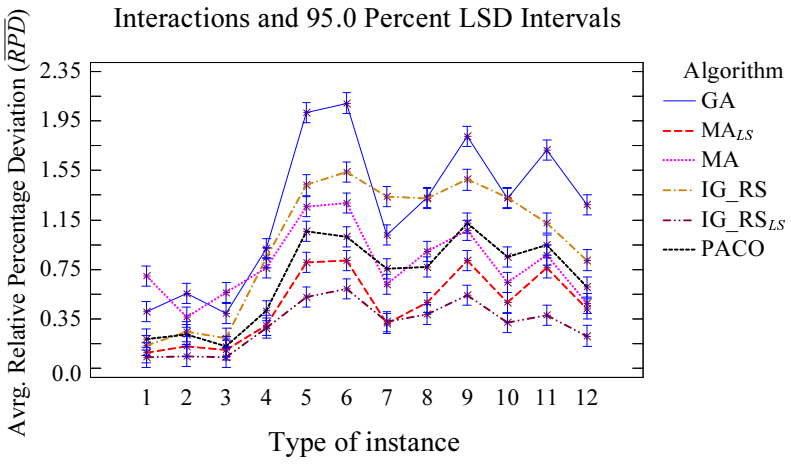

Figure 5: Plot of the average percentage deviation from best known solutions for the interaction between the type of algorithm and the type of instance. Instance set SDST10, $t = 90$ and makespan criterion.

In this plot, we see that in the easiest instances the algorithm's performances are closer together than in the hardest instances. According to the results, the instances with the largest deviations from the best known solutions are types 5, 6, 9 and 11, which correspond to 50×10,

50×20, 100×20 and 200×20. In these types, the differences among algorithms are much more acute. In all these cases we can see a clear lead for IG_RS$_{LS}$ followed by MA$_{LS}$ and sometimes by MA or PACO.

More or less the same conclusions as for instance set SDST10 can be drawn for the other 3 experiments with some minor exceptions. In Figure 6 we show the interaction between the factors algorithm and t for SDST125.

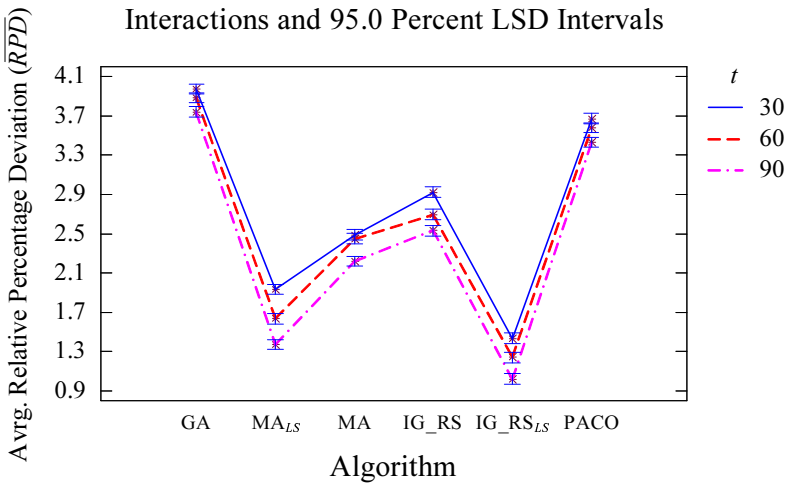

Figure 6: Plot of the average percentage deviation from best known solutions for the interaction between the type of algorithm and the different values of t for the stopping criterion. Instance set SDST125 and makespan criterion.

As it has been mentioned, the differences among the different values of t are more marked on this set (note that although in the graph the three lines appear to be closer together, the y-axis shows that the differences in the *AVRPD* values are greater than in the SDST10 experiment). Two striking differences from the SDST10 experiment can be pointed out. First IG_RS's performance is considerably better and almost comparable to that of MA. From the plot it seems that IG_RS with $t = 90$ is comparable to MA with $t = 60$. The second striking fact is the acute deterioration of the PACO algorithm, which in this case is far worse than all algorithms except GA. This experiment also indicates that a more in-depth adaptation of PACO to the SDST-FSP-C_{max} would be required.

As a first important observation we can say that the Iterated Greedy algorithms, especially the version with local search, are very robust since they resulted to be the best algorithms for the SDST-FSP-C_{max} in all four instance sets and they were also state-of-the-art algorithms for the regular flowshop problem (see Ruiz and Stützle 2006). Note that the adaptations needed for the SDST flowshop are small and even in this case the performance is still top notch.

4.2 Experimental results for the SDST-FSP-WT

Now we proceed to test the proposed algorithms as well as other existing methods for the total weighted tardiness objective. As in the previous section, we first need a benchmark set. We have chosen to take the already described SDST10,...,SDST125 sets and to augment them by including weights and due dates. The four new benchmark sets will be called DD_SDST10, ..., DD_SDST125. The weights for all n jobs (w_j) are drawn from a uniform $U[1, 10]$ distribution, which is also done for other benchmark sets for scheduling problems involving tardiness objectives

(Congram et al., 2002). For obtaining the due dates we use a similar approach to that of Hasija and Rajendran (2004). In that work, the authors consider the regular flowshop with total tardiness objective and they assign tight due dates in their experiments. In order to determine the due dates with sequence dependent setup times, we follow the following steps:

1. Calculate the total processing time on all the m machines for each job:

$$\forall j, j \in N,\ P_j = \sum_{i=1}^{m} p_{ij}$$

2. Calculate the average setup time for all possible following jobs and sum it for all machines:

$$\forall j, j \in N,\ S_j = \sum_{i=1}^{m} \frac{\sum_{k=1, k \neq j}^{n} S_{ijk}}{n-1}$$

3. Assign due date for each job:

$$\forall j, j \in N,\ d_j = (P_j + S_j) \times (1 + random \cdot 3)$$

where *random* is a real random number uniformly distributed in $[0, 1]$.

This method of generating due dates results in very tight to relatively tight due dates depending on the actual value of *random* for each job, i.e., if *random* is close to 0, then the due date of the job is going to be really tight as it would be more or less the sum of its processing times and average setup times on each machine. As a result, the job will have to be sequenced very early to avoid any tardiness. These 480 augmented instances as well as the best known solutions can be downloaded from `http://www.upv.es/gio/rruiz`.

As said in Section 2, we are only aware of three existing algorithms for the SDST-FSP-WT. We test the simulated annealing of Parthasarathy and Rajendran (1997a) and Parthasarathy and Rajendran (1997b) which we refer to as SA_PR and the heuristic as well as the version with the improvement scheme proposed of Rajendran and Ziegler (1997) and Rajendran and Ziegler (2003). We will refer to these two latter algorithms as HA and HA_IS2. We also test the proposed IG_RS and IG_RS$_{LS}$ algorithms as well as the previously tested genetic algorithms GA, MA and the new MA$_{LS}$. We have refrained from including other existing methods as, for example, PACO since it was proposed for a different objective. Therefore, comparing such an algorithm would be unfair since probably a totally reworked method, and not just an adaptation, should be compared.

It has to be mentioned that, for IG_RS, IG_RS$_{LS}$, GA, MA and MA$_{LS}$ the initialization procedure is changed from NEH_RMB to NEH_EWDD. In order to validate this decision, we carry out an experiment in which the original NEH and the NEH_EWDD are compared. We also test the heuristic method HA. The results for all instance sets can be seen in Table 3.

We can see that the *AVRPD* values are very large for all heuristics, instance sets and sizes. From the results, the heuristic HA, specifically designed for this problem and this objective, yields across all instances the worst results in most of the cases. When compared to the original NEH, HA obtains slightly better results only in instance sets DD_SDST100 and DD_SDST125. However, this poor performance of HA compared to NEH is mainly due to its very poor performance on the instances of size 20×20, while on the other instance sizes it is occasionally slightly better than NEH. In any case, the modified NEH_EWDD gives much better results than either

Instance	HA	NEH	NEH_EWDD	HA	NEH	NEH_EWDD
	DD_SDST10 Instances			DD_SDST50 Instances		
20×5	78.72	83.00	25.53	55.31	47.74	22.51
20×10	676.49	650.10	218.41	454.86	486.43	169.72
20×20	1532.80	736.68	492.98	1571.25	1048.81	483.75
50×5	20.36	26.00	8.99	19.37	23.96	13.29
50×10	52.04	51.97	22.02	39.15	34.66	17.48
50×20	218.85	222.13	93.04	78.37	78.90	32.26
100×5	12.46	14.86	6.82	11.95	14.67	8.93
100×10	20.93	23.04	9.89	18.12	17.89	10.56
100×20	47.46	47.72	19.78	28.12	30.25	13.92
200×10	11.15	11.11	4.83	8.36	9.84	4.91
200×20	18.10	16.59	6.48	11.03	11.50	5.37
500×20	9.33	5.87	1.37	5.77	5.14	1.93
Average	224.89	157.42	75.84	191.80	150.82	65.39
	DD_SDST100 Instances			DD_SDST125 Instances		
20×5	52.77	63.97	20.34	48.52	56.46	20.08
20×10	166.49	184.10	60.89	99.97	88.01	40.09
20×20	1130.55	1518.88	396.33	1522.99	1503.37	741.14
50×5	20.87	25.51	15.95	19.49	28.10	19.20
50×10	23.43	29.30	17.29	22.13	29.95	18.54
50×20	54.65	73.44	26.83	47.98	49.78	24.43
100×5	11.74	16.48	11.04	10.95	17.88	10.21
100×10	12.56	16.14	10.63	11.08	17.67	10.43
100×20	19.82	24.77	12.25	18.58	20.72	11.81
200×10	5.38	9.04	5.07	5.12	11.23	6.52
200×20	7.35	10.17	4.89	6.41	9.15	4.18
500×20	2.96	5.22	2.40	2.62	6.25	3.86
Average	125.72	164.75	48.66	151.32	153.21	75.87

Table 3: Average percentage increase over the best known solutions for the heuristic algorithms on the SDST-FSP-WT for all four instance sets.

of HA or NEH. As a matter of fact, over the 480 instances, NEH_EWDD gives better result than HA in 420 cases and better results than the original NEH in 470 cases. Hence, there is no need of an statistical analysis to ascertain that NEH_EWDD is the best heuristic among these three. However, this performance comes at a computational cost. The CPU times measured on an Athlon XP 1600+ with 512 MBytes of main memory, given in Table 4, needed by NEH and NEH_EWDD are very large compared to those of HA. (Since the computation times do not depend on the relative duration of the setups to processing times and they are similar for the four groups of instances, Table 4 gives only the times for set DD_SDST125.)

As shown, HA is a very fast method, while both, NEH and NEH_EWDD, have very large CPU times especially for the largest instances. Given these CPU times, we might expect a degradation of performance for the algorithms on the 500×20 instances. For the remaining cases, especially for cases where $n \leq 100$, the clear advantage of NEH_EWDD over HA would most probably prove beneficial since the CPU times needed for these cases are very small.

For testing the algorithms we use the same computer and methodology for the experiments as in the tests carried out for the SDST-FSP-C_{max}. In this case, every algorithm is run 10 independent times for a maximum CPU time given by $(n \cdot m/2) \cdot 180$. As commented previously,

Instance	HA	NEH	NEH_EWDD
20×5	0.00	0.00	0.00
20×10	0.00	0.00	0.00
20×20	0.00	0.00	0.00
50×5	0.00	0.01	0.01
50×10	0.00	0.02	0.02
50×20	0.00	0.04	0.04
100×5	0.00	0.08	0.08
100×10	0.01	0.16	0.17
100×20	0.01	0.32	0.34
200×10	0.03	1.59	1.60
200×20	0.04	6.52	6.56
500×20	0.34	172.39	172.46
Average	0.04	15.09	15.11

Table 4: Average CPU times (in seconds) for the heuristic algorithms on the SDST-FSP-WT for instance set DD_SDST125.

calculating the total weighted tardiness objective is much more expensive than the C_{max} objective. Additionally, the initialization procedure needs a large CPU time for the large instances and hence this longer CPU time stopping criterion.

The results for instance sets DD_SDST10 and DD_SDST50 are in Table 5, those for the sets DD_SDST100 and DD_SDST125 in Table 6.

These results show that the previous existing, SDST-FSP-WT specific algorithms SA_PR and HA_IS2 yield the worst solution qualities. In some cases, like instance sizes 20×20 for group DD_SDST125, SA_PR gives an *AVRPD* of more than 40%. Considering that all compared methods run for the same CPU time, SA_PR and HA_IS2 are clearly not the best options. However, it is worth mentioning that HA_IS2 gives the best results in all instance sets for the largest instances of 500×20. The previous analysis of the heuristics indicates that HA_IS2 benefits in this case from the fast HA initialization.

Overall, the third worst performing algorithm is GA, although in most cases it provides better results than HA_IS2, which uses an intensive form of local search. Interestingly, the performance of MA is in some cases considerably worse than that of the GA method, as, for example, in the instances 20×20 in the group DD_SDST50. While this result might seem counter-intuitive, it is clear that the hybridization with the curtailed local search in the MA algorithm is not effective enough. MA_{LS} provides results that are considerably better than those of MA. This demonstrates the usefulness of the local search used in our work.

Among the two, newly proposed methods, IG_RS shows very good performance, in some cases even better than MA_{LS}. On most instances, however, IG_RS_{LS} is the best performing algorithm. On average, and depending on the instance group, it gives *AVRPD* values that are between 9.3% and 27% lower than the second best algorithm MA_{LS}.

We again carry out four analyses of experiments with the ANOVA technique in the same way as described before. However, in this case we control only two factors, namely the type of algorithm and the type of instance, since the parameter t is not used to vary the computation times. As for the SDST-FSP-C_{max}, also for the SDST-FSP-WT the overall results of the four experiments are rather similar and, hence, we illustrate the main findings giving some example results. Figure 7 shows the LSD means plot for the factor type of algorithm in the experiment DD_SDST100.

Instance	GA	MA	MA_{LS}	IG_RS	IG_RS_{LS}	SA_PR	HA_IS2
DD_SDST10 Instances							
20×5	0.92	2.16	0.00	0.00	0.00	3.53	4.61
20×10	1.60	8.34	0.00	0.00	0.00	9.90	6.43
20×20	1.88	0.00	0.00	0.00	0.00	22.45	8.09
50×5	2.83	1.44	1.17	2.70	0.77	5.17	4.56
50×10	5.39	1.99	1.63	3.62	1.22	8.13	7.88
50×20	10.80	4.28	2.60	8.54	1.51	16.37	20.79
100×5	2.83	2.40	1.95	3.52	1.39	4.37	3.04
100×10	3.91	2.80	2.50	4.17	2.12	5.53	4.93
100×20	6.44	4.23	3.84	5.95	3.41	7.72	8.38
200×10	1.94	2.61	1.86	2.29	1.35	3.50	1.35
200×20	2.06	4.90	3.47	2.29	2.30	3.44	2.34
500×20	0.78	1.37	0.80	0.58	0.66	2.76	0.11
Average	3.45	3.04	1.65	2.80	1.23	7.74	6.04
DD_SDST50 Instances							
20×5	1.45	6.96	0.00	0.02	0.00	5.92	7.20
20×10	5.28	13.87	0.09	0.15	0.15	12.58	24.90
20×20	0.43	30.78	0.00	0.00	0.00	4.79	1.68
50×5	5.42	2.19	2.00	3.77	1.79	7.59	7.31
50×10	5.83	2.02	2.24	3.79	2.41	8.42	9.13
50×20	6.75	2.53	2.08	4.69	2.80	10.95	12.97
100×5	4.47	3.14	2.66	4.82	2.14	6.70	4.19
100×10	4.65	3.09	2.61	4.59	2.51	6.65	6.25
100×20	5.42	3.39	3.15	5.22	3.29	7.12	6.06
200×10	1.95	2.56	1.71	2.37	1.07	3.65	1.12
200×20	1.61	4.10	2.70	1.95	1.61	3.10	1.51
500×20	1.43	1.93	1.42	1.04	0.95	3.76	0.00
Average	3.72	6.38	1.72	2.70	1.56	6.77	6.86

Table 5: Average relative percentage deviation from the best known solutions for the algorithms for the SDST-FSP-WT on instance sets DD_SDST10 and DD_SDST50 with the termination criterion set at $(n \cdot m/2) \cdot 180$ milliseconds maximum CPU time.

Instance	GA	MA	MA_{LS}	IG_RS	IG_RS_{LS}	SA_PR	HA_IS2
DD_SDST100 Instances							
20×5	3.99	2.89	0.07	0.07	0.40	8.24	9.77
20×10	2.67	6.53	0.09	0.11	0.10	13.21	18.28
20×20	2.80	2.49	0.00	0.00	0.00	18.17	40.94
50×5	7.57	3.89	3.38	5.83	3.46	11.46	10.75
50×10	7.80	3.23	3.00	5.34	2.93	10.77	9.76
50×20	8.08	2.78	2.57	5.06	2.36	11.34	13.15
100×5	5.22	3.71	3.24	5.91	2.76	8.40	5.44
100×10	4.76	3.52	2.80	5.45	2.57	7.58	5.17
100×20	5.35	3.39	3.13	5.27	3.29	7.72	6.43
200×10	2.60	3.55	2.34	2.60	1.27	4.78	1.63
200×20	1.96	4.42	3.20	2.33	1.60	3.77	1.61
500×20	2.38	2.89	2.37	2.26	0.63	6.38	0.14
Average	4.60	3.61	2.18	3.35	1.78	9.32	10.26
DD_SDST125 Instances							
20×5	3.68	5.54	0.41	0.15	0.40	10.22	14.82
20×10	3.91	6.28	0.10	0.00	0.04	10.80	15.12
20×20	3.01	11.57	0.00	0.00	0.00	42.38	14.08
50×5	8.36	4.07	3.75	7.20	3.62	12.67	11.43
50×10	8.01	3.63	3.49	6.25	3.79	11.16	10.50
50×20	7.83	2.63	2.75	5.56	2.74	11.45	10.70
100×5	5.01	3.73	2.87	5.58	2.01	9.27	4.78
100×10	4.69	3.38	2.81	5.30	2.52	7.88	5.35
100×20	4.68	3.07	2.61	4.64	2.79	6.97	5.97
200×10	3.10	4.38	2.97	3.46	0.99	6.00	1.23
200×20	2.08	3.48	3.10	1.84	1.17	4.10	1.35
500×20	3.20	3.86	3.16	3.13	0.45	7.43	0.32
Average	4.80	4.63	2.34	3.59	1.71	11.69	7.97

Table 6: Average relative percentage deviation from the best known solutions for the algorithms for the SDST-FSP-WT on instance sets DD_SDST100 and DD_SDST125 with the termination criterion set at $(n \cdot m/2) \cdot 180$ milliseconds maximum CPU time.

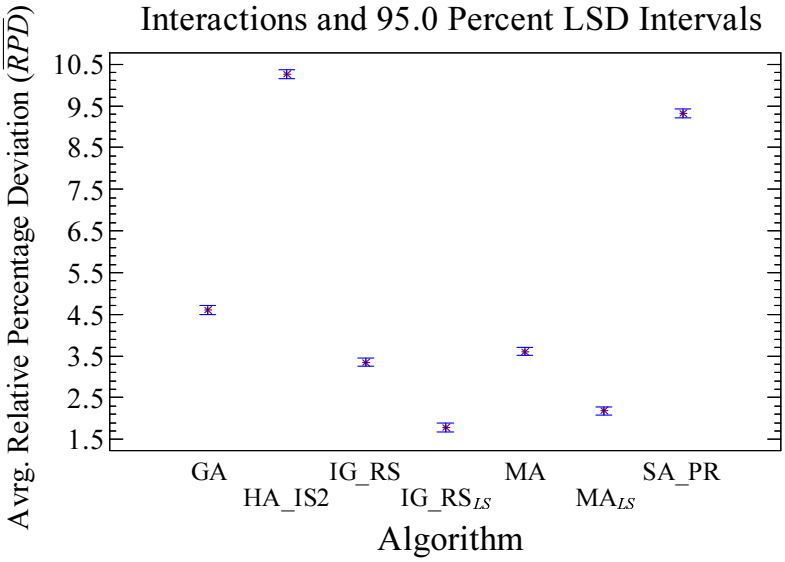

Figure 7: Plot of the average percentage deviation from best known solutions for the type of algorithm factor. Instance set SDST100 and total weighted tardiness criterion.

We see that there are statistically significant differences among all algorithms with the only exception of MA and IG_RS that are, at the 95% confidence level, statistically equivalent. A finer examination and a close look at Table 6 shows that there are differences depending on the instance type. Figure 8 depicts a more precise example where we examine the differences among the algorithms for a same instance group, here type 11, i.e., instances of size 200×20.

In this case, we have that the differences observed in the $AVRPD$ values are statistically significant with the only exception of IG_RS$_{LS}$ and HA_IS2, which from Table 6 have $AVRPD$ values of 1.60 and 1.61 respectively. Thanks to the number of replicates and the 480 total instances, the LSD intervals that result from the experiments are fairly narrow, which indicates that most of the observed differences in Tables 5 and 6 are statistically significant.

Similarly to the SDST-FSP-C_{max} case, we have just examined the results averaged across all instance types. Figure 9 shows, for set SDST10, the interaction between the type of instance and algorithm factors. As shown in the results, the algorithms HA_IS2 and SA_PR show a poor performance for the smaller instances with improving results as the size of the instances grows.

In summary, also for the SDST-FSP-WT, especially the proposed hybrid version of the iterated greedy appears to give excellent results. Only for the very largest instances, another algorithm that is specifically designed for the SDST-FSP-WT gives slightly better results; however, this effect may be mainly due to the relatively slow initialization of our current algorithm and IG_RS$_{LS}$ is clearly the best algorithm on all the other instances.

5 Conclusion and future research

In this article, we have presented two new Iterated Greedy (IG) algorithms for SDST flowshop problems that were tackled under two of the most widely used and relevant objective functions, the minimization of the maximum completion time or makespan and the minimization of the total weighted tardiness. The first algorithm, called IG_RS, works by iterating over greedy construction heuristics, in this case based on the well known NEH heuristic from Nawaz et al.

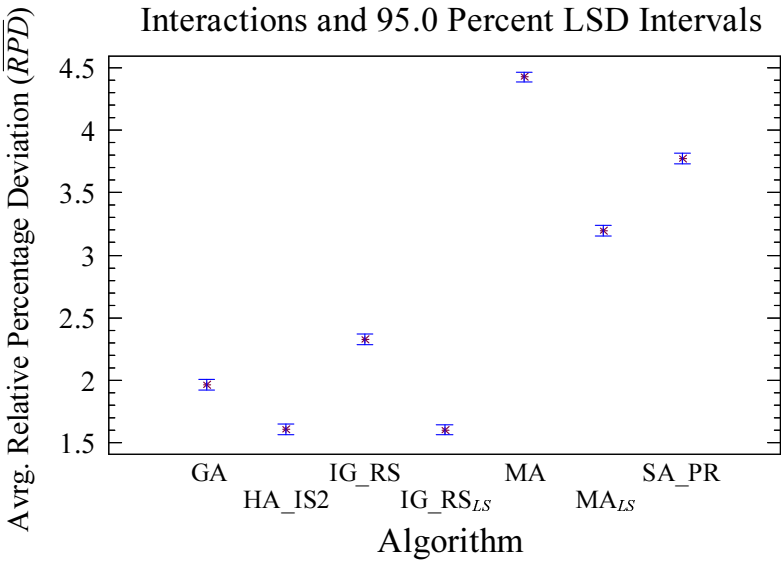

Figure 8: Plot of the average percentage deviation from best known solutions for the type of algorithm factor. Instance set SDST100, instance type 11 (200×20) and total weighted tardiness criterion.

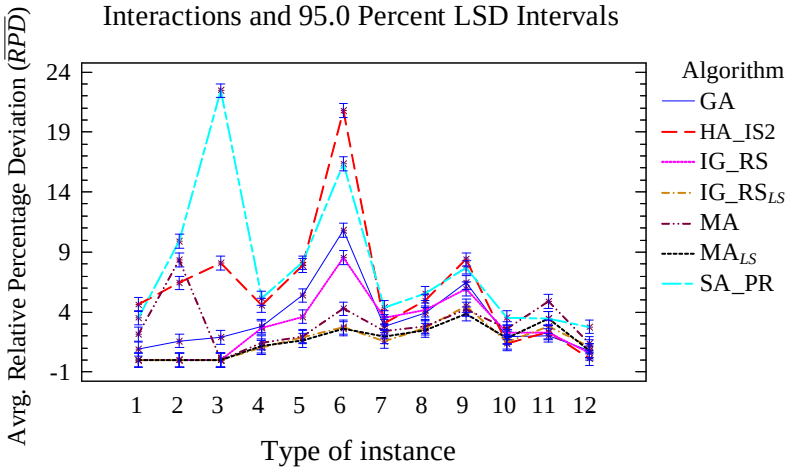

Figure 9: Plot of the average percentage deviation from best known solutions for the interaction between the type of algorithm and the type of instance. Instance set SDST10 and total weighted tardiness criterion.

(1983), and by applying two phases, named *destruction*, where some jobs are removed from the incumbent sequence, and *construction*, where the greedy heuristic is applied to reconstruct the sequence and to reinsert the jobs that were previously removed. The second, IG_RS$_{LS}$, is a straightforward extension of IG_RS by including an additional local search phase, in which a simple descent algorithm is applied. For both algorithms, extensive experiments and statistical analyses have been carried out.

The experimental results establish IG_RS$_{LS}$ as a new state-of-the-art algorithm for the scheduling problems tackled, while IG_RS was shown to be the best algorithm that does not use an explicit local search phase. This excellent performance of the iterated greedy algorithms is very noteworthy because of two main reasons. First and foremost, the iterated greedy algorithms have a very simple structure, they are easy to understand, and also very easy to implement. This makes them preferable over other, much more complex algorithms—even if the performance would be the same. Additionally, the IG algorithms applied here were based on a successful IG algorithm for the usual flowshop problem under the makespan criterion (PFSP) Ruiz and Stützle (2006)—a first proof that IG algorithms may easily be extended to related problems maintaining their high performance. Second, these results provide some first evidence that the Iterated Greedy method is an excellent candidate for tackling complex scheduling environments as they are encountered in real-world environments. In fact, the FFSP, to which IG was first applied Ruiz and Stützle (2006), is considered as being a rather unrealistic environment concerning real applications (Dudek et al., 1992).

There are mainly two interesting lines of future research, those concerning algorithmic issue and more realistic scheduling environments, respectively. In the first direction, it would be interesting to examine for which classes of combinatorial problems the IG method can be highly competitive. Certainly, the availability of high-performing construction methods are an essential ingredient, but also other factors may be central to the performance of IG algorithms. In the second direction, we would like to test IG based algorithms on other features of flowshop scheduling environments like no-wait, availability of parallel machines, other objectives etc. This direction is important to identify with which characteristics of scheduling problems the IG algorithms can deal effectively. The ultimate goal is to apply IG to more complex scheduling environments, both with the consideration of more realistic constraints as well as with other objectives. Clear examples of this are flowshops or jobshops where the production floor is divided up into stages and at each stage several identical or even unrelated parallel machines are present. These environments, the so-called hybrid shops, are of great importance in practice and IG methods may be excellent candidates for tackling them.

Acknowledgments

Rubén Ruiz is partly funded by the Spanish Department of Science and Technology (research project ref. DPI2004-06366-C03-01). Thomas Stützle acknowledges support of the Belgian FNRS, of which he is a research associate.

References

Congram, R. K., Potts, C. N., and van de Velde, S. (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67.

Corwin, B. D. and Esogbue, A. O. (1974). Two machine flow shop scheduling problems with

sequence dependent setup times: A dynamic-programming approach. *Naval Research Logistics*, 21(3):515–524.

Das, S. R., Gupta, J. N. D., and Khumawala, B. M. (1995). A savings index heuristic algorithm for flowshop scheduling with sequence-dependent set-up times. *Journal of the Operational Research Society*, 46(11):1365–1373.

Dudek, R., Panwalker, S., and Smith, M. (1992). The lessons of flowshop scheduling research. *Operations Research*, 40(1):7–13.

Gupta, J. N. D. (1986). Flowshop schedules with sequence dependent setup times. *Journal of the Operations Research Society of Japan*, 29(3):206–219.

Gupta, J. N. D. and Darrow, W. P. (1986). The two-machine sequence dependent flowshop scheduling problem. *European Journal of Operational Research*, 24(3):439–446.

Hasija, S. and Rajendran, C. (2004). Scheduling in flowshops to minimize total tardiness of jobs. *International Journal of Production Research*, 42(11):2289–2301.

Hoos, H. H. and Stützle, T. (2004). *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA.

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

Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). Iterated local search. In Glover, F. and Kochenberger, G. A., editors, *Handbook of metaheuristics*, pages 321–353, Boston. Kluwer Academic Publishers.

Marchiori, E. and Steenbeek, A. (2000). An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. In Cagnoni, S. et al., editors, *Real-World Applications of Evolutionary Computing, EvoWorkshops 2000*, volume 1803 of *Lecture Notes in Computer Science*, pages 367–381. Springer Verlag, Berlin, Germany.

Nawaz, M., Enscore, J. E. E., and Ham, I. (1983). A heuristic algorithm for the m machine, n job flowshop sequencing problem. *Omega-International Journal of Management Science*, 11(1):91–95.

Osman, I. H. and Potts, C. N. (1989). Simulated annealing for permutation flowshop scheduling. *Omega-International Journal of Management Science*, 17(6):551–557.

Parthasarathy, S. and Rajendran, C. (1997a). An experimental evaluation of heuristics for scheduling in a real-life flowshop with sequence-dependent setup times of jobs. *International Journal of Production Economics*, 49(3):255–263.

Parthasarathy, S. and Rajendran, C. (1997b). A simulated annealing heuristic for scheduling to minimize mean weighted tardiness in a flowshop with sequence-dependent setup times of jobs - a case study. *Production Planning & Control*, 8(5):475–483.

Pinedo, M. (2002). *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, Upper Saddle, N.J, second edition.

Rajendran, C. and Ziegler, H. (1997). A heuristic for scheduling to minimize the sum of weighted flowtime of jobs in a flowshop with sequence-dependent setup times of jobs. *Computers & Industrial Engineering*, 33(1-2):281–284.

Rajendran, C. and Ziegler, H. (2003). Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. *European Journal of Operational Research*, 149(3):513–522.

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(2):426–438.

Ruiz, R. and Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494.

Ruiz, R., Maroto, C., and Alcaraz, J. (2005). Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. *European Journal of Operational Research*, 165(1):34–54.

Ruiz, R. and Stützle, T. (2006). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*. Accepted for publication.

Ríos-Mercado, R. Z. and Bard, J. F. (1998a). Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. *Computers & Operations Research*, 25(5):351–366.

Ríos-Mercado, R. Z. and Bard, J. F. (1998b). Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, 110(1):76–98.

Ríos-Mercado, R. Z. and Bard, J. F. (1999a). A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. *IIE Transactions*, 31(8):721–731.

Ríos-Mercado, R. Z. and Bard, J. F. (1999b). An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup times. *Journal of Heuristics*, 5(1):53–70.

Simons, Jr, J. V. (1992). Heuristics in flow shop scheduling with sequence dependent setup times. *Omega-International Journal of Management Science*, 20(2):215–225.

Srikar, B. N. and Ghosh, S. (1986). A MILP model for the n-job, m-stage flowshop with sequence dependent set-up times. *International Journal of Production Research*, 24(6):1459–1474.

Stafford, Jr, E. F. and Tseng, F. T. (1990). On the Srikar-Ghosh MILP model for the $n \times m$ SDST flowshop problem. *International Journal of Production Research*, 28(10):1817–1830.

Stützle, T. (1998). Applying iterated local search to the permutation flow shop problem. Technical Report AIDA–98–04, FG Intellektik, FB Informatik, TU Darmstadt.

Taillard, E. (1990). Some efficient heuristic methods for the flow-shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285.

Tseng, F. T. and Stafford, Jr, E. F. (2001). Two MILP models for the $n \times m$ SDST flowshop sequencing problem. *International Journal of Production Research*, 39(8):1777–1809.