An Iterated Greedy Algorithm for the Flowshop Problem with Sequence Dependent Setup Times

Rubén Ruiz* Thomas Stützle†

*Departamento de Estadística e Investigación Operativa Aplicadas y Calidad,
Grupo de Investigación Operativa, Universidad Politécnica de Valencia
Camino de Vera S/N, 46021 Valencia, Spain
rruiz@eio.upv.es

†Computer Science Department, Darmstadt University of Technology
Hochschulstr. 10, 64283 Darmstadt, Germany
stuetzle@informatik.tu-darmstadt.de

1 Introduction

The flowshop problem (FSP) is one of the most thoroughly studied scheduling problems. In the FSP, a set $N = \{1, \ldots, n\}$ of n independent jobs has to be processed on a set $M = \{1, \ldots, m\}$ of m machines. Every job j, $j \in N$, requires a given fixed, non-negative processing time p_{ij} on every machine i, $i \in M$. In a flowshop, all n jobs are to be processed on the m machines in the same order, that is, the jobs follow the same flow in the shop starting from machine 1 and finishing on machine m. The criterion that is most commonly studied in the literature is the minimization of the total completion time, also called makespan C_{max}. A common simplification in the FSP is to avoid *job passing* in the sequence; that is, the processing sequence on the first machine is maintained throughout the remaining machines. The resulting problem is the permutation flowshop problem (PFSP).

Despite the enormous research effort on the FSP, there have been significant objections to the practical relevance of the (P)FSP [1]. Therefore, extensions of the basic FSP environment recieve increasing attention in the literature. One such extension is to consider sequence dependent setup times (SDST). Setup times involve 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 times, that is the setup operations can be done before the product arrives to the machine, and when they are sequence dependent, that is, the amount of setup time depends on the machine, on the job that the machine was processing and on the job that comes next. A clear example comes from 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 s_{ijk} is the known, fixed and non-negative setup time of machine i when producing job k after having produced job j.

The permutation SDST flowshop problem with C_{max} objective (SDST-PFSP) has been shown to be $\mathcal{NP}$-complete by even when $m = 1$ [2]. Not only in theory, but also in practice, the SDST-PFSP appears to be very hard. Several attempts solving it have been made with exact algorithms, including integer programming [18] and specialized branch-and-bound methods [9, 11]. However, most of these algorithms were limited at the time to instances of size around $n = 10$, which basically could also be solved within reasonable computation times by enumeration. Therefore, a number of stochastic local search (SLS) algorithms [3] have been applied to this problem, ranging from simple heuristics [14], adaptations of the well-known NEH heuristics for the PFSP to the SDST environment [10] to GRASP and memetic algorithms [10, 12]. For an overview of algorithmic approaches to the SDST-PFSP, we refer to [12].

Recently, a new SLS method called Iterated Greedy (IG) has shown state-of-the-art performance for the very competitive field of PFSP with minimization of the makespan as an objective [13]. In this article, we show that IG also results in a highly effective method for the SDST-PFSP using rather straightforward adaptations. A description of the IG algorithm for the SDST-PFSP is given in the next section, a sketch of the experimental results is presented in Section 3 and we conclude in Section 4.

2 IG for the SDST-PFSP

IG is remarkably simple and works by iteratively applying two phases, named *destruction*, were some elements are eliminated from a current solution, thus obtaining a *partial* solution, and *construction*, where elements are inserted into a partial solution until a complete one is obtained again. An acceptance criterion is applied to the reconstructed candidate solution to decide if it replaces the incumbent solution. Then, the process is iterated until some stopping criterion is met. IG is closely related to other Stochastic Local Search methods, in particular to Iterated Local Search (ILS) [5]. The applications of the IG method so far are limited and mostly for the set covering problem [4, 6]. Recently, excellent results have been obtained with such an approach for the PFSP [13], where it yielded results that were superior to those from several much more complex algorithms. Here, we study the performance of a direct extension of this earlier IG algorithm to the SDST-PFSP and compare it to the current state-of-the-art algorithms for that problem. Next, we shortly describe IG method.

Algorithm initialization. The initial sequence, from which the main loop of the IG method starts, is constructed using the NEHT_RMB heuristic, which was proposed by Rios-Mercado and Bard [10]; this heuristic is an adaptation of the well-known NEH heuristic to the SDST-PFSP. The same speed-ups as proposed for the NEH in [16] can also be transferred to the SDST-PFSP and, hence, the heuristic has a computational complexity of $\mathcal{O}(n^2m)$.

Destruction, construction and acceptance criterion The destruction step starts from some initial sequence π and removes from it d jobs, which are chosen at random and without

repetition. This process creates two subsequences. The first is π_R with d jobs, which contains the removed jobs in the order in which they were extracted from π; the second is π_D with $n-d$ jobs, which is π without the removed jobs. Starting from these two subsequences, the construction step reinserts all jobs of π_R into π_D by applying the NEH heuristic (using the speed-ups of Taillard [16]); that is, the first job of π_R, $\pi_{R(1)}$, is tentatively inserted in all possible $n-d+1$ positions of π_D and the best resulting partial sequence that includes $\pi_{R(1)}$ is kept. This process is iterated until π_R is empty and π_D is again a complete solution.

Another step in IG is to decide whether the reconstructed sequence is accepted or not as the incumbent solution for the next iteration. Here, we adopt a simulated annealing-like acceptance criterion with constant temperature as used in [7, 15, 13]. The constant temperature depends on the processing times, the number of jobs and machines and an adjustable parameter T:

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

Let $f(\pi_i)$ and $f(\pi)$ be the evaluation function values of the incumbent sequence and the reconstructed sequence, respectively. If $f(\pi) > f(\pi_i)$, then π is accepted as the new incumbent if $random \leq \exp\{f(\pi) - f(\pi_i)/\text{Temperature}\}$, where *random* is a random number uniformly distributed in $[0, 1]$; if $f(\pi) \leq f(\pi_i)$, it is accepted always.

Hybridization with local search. To further improve the performance of the IG algorithm, a simple iterative improvement algorithm can optionally be applied before the acceptance criterion decides from which solution the algorithm continues. Our local search is based on the insertion neighborhood, where for a sequence π all sequences are neighboured that can be obtained by removing one job in π and inserting it in a different position than it has in π. The algorithm examines the neighborhood for each job; if for a job an improvement results the best sequence among the n possible ones is chosen as the next one. The final IG algorithm with this optional local search step is given in Figure 1. In the following we refer with IG_{LS} to the IG algorithm with local search, while IG alone refers to the version without local search.

3 Experimental evaluation

Experimental tests with IG were done on a comprehensive set of benchmark instances. This set consists of the four instance sets that were used for testing algorithms in [12]. Each set is composed of the original 120 instances of [17], which are again 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 four benchmark sets we use differ in the setup times to processing time ratios. These ratios were taken as 10%, 50%, 100%, and 125%, resulting in the four sets SDST10, SDST50, SDST100, and SDST125 of 120 instances each. The instances can be downloaded from `http://www.upv.es/gio/rruiz`.

In the experimental study, the parameters of IG were set as $T = 0.5$ and $d = 4$, following [13] and some preliminary tests. We compare the two Iterated Greedy algorithms to the genetic (GA) and memetic algorithm (MA) of [12] and an adaptation to the SDST-PFSP of an ant colony optimization algorithm called PACO, which was originally proposed for the PFSP in

```
procedure IteratedGreedy_for_SDST_flowshop
   π := NEH_RMB;        % 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 f(π'') ≤ f(π) then                 % Acceptance Criterion
         π := π'';
         if f(π) < f(π_b) then              % check if new best permutation
            π_b := π;
         endif
      elseif (random ≤ exp{−(f(π'') − f(π))/Temperature}) then
         π := π'';
      endif
   endwhile
   return π_b
end
```

Figure 1: Iterated Greedy algorithm with the optional local search phase.

[8]. PACO and IG_{LS} use the above mentioned local search. MA used a weaker form of local search, and therefore we run a version of it, called MA_{LS} with the same local search as IG_{LS} to avoid that differences between the performance of IG_{LS} and MA_{LS} are only due to the local search algorithm. Recall that a total of 14 algorithms were tested in [12], including all SDST-PFSP specific heuristics and adaptations of PFSP algorithms; there, MA resulted to be by far superior to all other algorithms.

All experiments were run on an Athlon XP 1600+ processor (1400 MHz) with 512 MB of main memory using 10 independent trials that were stopped after a CPU time computed as $(n \cdot m/2) \cdot t$ milliseconds, where $t \in \{30, 60, 90\}$. As the response variable, the average relative percentage deviation over the best solution known is computed on each instance and these values are then again averaged across the instance classes under consideration. (The best known solutions can be downloaded from `http://www.upv.es/gio/rruiz`.) The results for instance sets SDST100 and SDST125 are given in Table 1. (Results on the other instance classes are mostly similar regarding the relative ranking of the algorithms; these have not been included due to space limitations.) Notice that results are averaged per combination of n and m and therefore each cell of the table shows the average result across 100 values. In Figure 2 we give additional details on the results including Least Significative Difference (LSD) intervals, which indicate the statistical significance between the results. In part 2(a), we compare the algorithms across all instances of a benchmark set and in part 2(b) this comparison is done in more detail considering each of the 12 different instance sizes (numbered according to increasing size).

Table 1: Average percentage increase over the best known upper bound for the methods evaluated with the termination criteria set at $(n \cdot m/2) \cdot t$ milliseconds maximum CPU time where $t = 30, 60$ and 90. Instance sets SDST100 and SDST125.

Instance	GA	MA	MA_{LS}	PACO	IG	IG_{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

The following observations can be made from these results. Firstly, the GA results in the lowest performance of the algorithms compared. In fact, it gives also significantly worse results than IG, the only other algorithm that does not use a local search phase. This fact establishes IG as the best performing algorithm for the SDST-PFSP without local search. Secondly, the iterative improvement algorithm used in this article, results in higher performing algorithms than the less intensive local search used in MA; this is indicated by the fact that MA_{LS} performs significantly better than its predecessor, MA. Thirdly, IG_{LS} performs significantly better than all competitors, including MA_{LS}. This is a very remarkable fact, since MA, the predecessor of MA_{LS}, was previously shown to be by far better than existing as well as adapted methods for the SDST flowshop with C_{max} objective [12]. In addition, IG_{LS} is a much simpler and more straightforward algorithm than MA or MA_{LS}. These facts establish IG_{LS} as a new state-of-the-art heuristic for the SDST-PFSP.

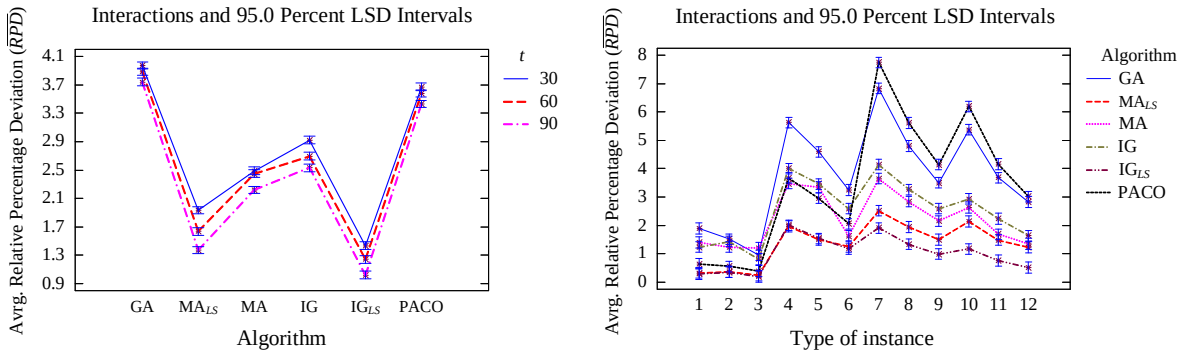

Figure 2: Left side: Plot of the average percentage deviation from best known solutions for the interaction between the type of algorithm and (left side) the different values of t for the stopping criterion and (right side) the instance class clustered according to size for $t = 90$. All the results are given for the instances set SDST125.

4 Conclusions

In this paper, we have described the application of an iterated greedy algorithm to the permutation flowshop problem with sequence dependent setup times and makespan criterion. The IG algorithm was previously shown to reach excellent performance on the regular permutation flowshop problem with makespan criterion, where it even could improve some best known solutions despite that fact that this problem was tackled by a large number of specialized, highly tuned algorithms. Interestingly, the high performance of IG for the PFSP also transfers to the extension with SDST, which is one of the extensions to make the regular PFSP a more realistic model. Additional results on the SDST-PFSP with total weighted tardiness objective indicate a similar level of performance for the IG method. This lets us believe that IG will also be very effective for other PFSP extensions and that it is an excellent candidate for algorithms that tackle real-world flowshop problems, since it combines two highly desirable properties of algorithms: simplicity and high performance.

References

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

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

[3] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search—Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, 2004.

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

[5] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[6] E. Marchiori and A. Steenbeek. An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. *Real-World Applications of Evolutionary Computing, Proceedings*, 1803:367–381, 2000.

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

[8] C. Rajendran and H. Ziegler. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, 155(2):426–438, 2004.

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

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

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

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

[13] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 2005. Accepted for publication.

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

[15] Thomas Stützle. Applying iterated local search to the permutation flow shop problem. Technical Report AIDA–98–04, FB Informatik, TU Darmstadt, August 1998.

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

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

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