Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 165 (2005) 34–54
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

Discrete Optimization

Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics

Rubén Ruiz *, Concepción Maroto, Javier Alcaraz

Departmento de Estadística e Investigación Operativa Aplicadas y Calidad, Universidad Politécnica de Valencia, Edificio I-3, Camino de Vera S/N, 46021, Valencia, Spain

Received 27 May 2003; accepted 8 January 2004
Available online 14 March 2004

Abstract

This paper deals with the permutation flowshop scheduling problem in which there are sequence dependent setup times on each machine, commonly known as the SDST flowshop. The optimisation criteria considered is the minimisation of the makespan or $C_{\max}$. Genetic algorithms have been successfully applied to regular flowshops before, and the objective of this paper is to assess their effectiveness in a more realistic and complex environment. We present two advanced genetic algorithms as well as several adaptations of existing advanced metaheuristics that have shown superior performance when applied to regular flowshops. We show a calibration of the genetic algorithm's parameters and operators by means of a Design of Experiments (DOE) approach. For evaluating the proposed algorithms, we have coded several, if not all, known SDST flowshop specific algorithms. All methods are tested against an augmented benchmark based on the instances of Taillard. The results show a clear superiority of the algorithms proposed, especially for the genetic algorithms, regardless of instance type and size.

Keywords: Scheduling; Genetic algorithms; Sequence dependent setup times; Local search

1. Introduction

The flowshop scheduling problem comprises a set $N = \{J_1, \ldots, J_n\}$ of n independent jobs that have to be processed on a set $M = \{M_1, \ldots, M_m\}$ of m different machines. Every job requires a known, deterministic and non-negative span of time for completion at each machine. This time is called processing time and it is denoted as p_{ij} $(i = 1, \ldots, m,\ j = 1, \ldots, n)$. In a flowshop, all jobs have to be processed sequentially on the machines and have the same processing order. The objective is to find an ordering of the jobs on the machines or *sequence* that optimises some given criteria. The most used criteria is the minimisation of the total completion time of the schedule, $F_{\max}$ often referred to as makespan or $C_{\max}$. The problem is classified as $n/m/F/F_{\max}$ (see Conway et al., 1967) or as $F//C_{\max}$ following the three parameter notation $(\alpha/\beta/\gamma)$ of Graham et al. (1979). Usually, only permutation schedules are considered, i.e. *job*

* Corresponding author. Tel.: +34-96-387-70-07x74946; fax: +34-96-387-74-99.
E-mail address: rruiz@eio.upv.es (R. Ruiz).

0377-2217/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2004.01.022

passing is not permitted and therefore, the processing sequence of the jobs is the same for all machines. In this case, there are $n!$ feasible schedules and the problem is then denoted as $n/m/P/F_{\max}$ or as $F/prmu/C_{\max}$ (see Pinedo, 2002).

An interesting and realistic variation of the permutation flowshop problem (PFSP) is when jobs have sequence dependent setup times on machines, that is, there is a significant setup time on each machine before processing the next job that depends on the job currently being processed. Thus, we will denote S_{ijk} as the known, deterministic and non-negative time span required for setting up machine i when job j is scheduled immediately before job k. When these setup times are specifically considered the problem is noted as $F/S_{ijk}, prmu/C_{\max}$ and commonly referred to as SDST flowshop. This scheduling problem was shown to be $\mathcal{NP}$-complete by Gupta (1986) so it is unlikely that a polynomial algorithm exists for solving the problem in its general form.

The SDST flowshop is applicable to many real cases where multi-purpose machines are used to manufacture different products. A clear example comes from the paper cutting industry where the paper cutting machines need to be adjusted when changing from one cutting batch to another. Another example can be obtained from the ceramic tile manufacturing sector. Ceramic tiles are produced in identical processing lines composed of several processes; molding press, dryer, glazing line, kiln, quality control and finally, packing and delivery. If a production line is setup for a given product, for example a 33×33 cm black glazed ceramic tile, changing to a 45×45 cm white glazed ceramic tile it will need significant setup times to change the molds in the molding press and to clean and prepare the glazing line for the white glaze. However, changing to a 33×33 cm yellow glazed tile will only require cleaning in the glazing line, as the molding press is already setup.

This paper tests two new advanced genetic algorithms (GA) for the SDST flowshop that were shown to be superior to other existing approaches in the regular flowshop by previous research work (see Ruiz et al., 2003). The first GA includes specific problem domain knowledge and some original procedures and operators that we designed. This includes four new types of crossover operators that have shown to be superior to other crossover operators tested, and new generational and restarting schemes. The second GA applies a local search based in an insertion neighbourhood in an improvement phase. We also propose the adaptation of several metaheuristics, which have proven to be competitive for the regular flowshop problem, for application to the SDST flowshop. All proposed algorithms are compared against several specific techniques proposed for the SDST flowshop. For the tests we used the standard benchmark of Taillard (1993), which is composed of 120 different problem instances ranging from 20 jobs and five machines to 500 jobs and 20 machines. For the consideration of sequence dependent setup times we have used an augmented version of this well known benchmark with setup times in all machines by Vallada et al. (2003).

The rest of the paper is organized as follows: in Section 2 we present a literature survey of the existing methods for the SDST flowshop. Section 3 deals with the tested genetic algorithm. In Section 3.5, the proposed algorithm is finely calibrated by means of the Design of Experiments (DOE) technique. In Section 4, the proposed genetic algorithm is hybridized with a local search procedure. Section 5 provides details on the metaheuristics that have been adapted for application to the SDST flowshop. An extensive and comprehensive evaluation of both proposed genetic algorithms with the adapted metaheuristics and other existing algorithms for the SDST flowshop is given in Section 6. Finally, Section 7 provides some conclusions on the study and some interesting future research work.

2. Literature review

Since Johnson's (1954) pioneering work on the two machine regular permutation flowshop, a wealth of research has been conducted in both exact and heuristic methods for the PFSP. Due to the $\mathcal{NP}$-completeness of the PFSP (see Garey et al., 1976), researchers have mainly focused on the development of effective heuristics and metaheuristics. Salient heuristics are due to Campbell et al. (1970) (also called CDS method) and the

well-known NEH heuristic by Nawaz et al. (1983). Some noteworthy metaheuristics have also been proposed, for example the simulated annealing by Osman and Potts (1989), the tabu search of Widmer and Hertz (1989) and the genetic algorithm of Reeves (1995). For a recent review and evaluation of PFSP heuristics and metaheuristics, the reader can refer to Ruiz and Maroto (2004).

Compared to the work and achievement obtained in the PFSP problem, the SDST flowshop has attracted much less attention. Corwin and Esogbue (1974) formulated a dynamic programming approach for a special case of the SDST flowshop where there are only two machines and sequence dependent setup times exist only in the second machine. Gupta (1975) presented a lexicographic search algorithm for a generalized flowshop problem where sequence dependent setup times can also be considered and where the objective function is the minimisation of the total opportunity cost. Gupta and Darrow (1986) showed several heuristics for the case of the SDST flowshop with two machines. Szwarc and Gupta (1987) also developed an approximate algorithm based on the multi-sort technique for the two machine case where the setup times are considered to be additive. Srikar and Ghosh (1986) developed an MILP model for the SDST flowshop and they were able to solve instances of up to six jobs and six machines. Later, Stafford and Tseng (1990) corrected the previous model and were able to solve instances of up to seven jobs and five machines. Simons (1992) proposed two SDST flowshop sequencing heuristics, named TOTAL and SETUP which are based on a Travelling Salesman Problem (TSP) formulation of the SDST flowshop that is solved with the well known Vogel's approximation method. Das et al. (1995) showed an algorithm for the SDST flowshop based on a savings index. The proposed algorithm dynamically calculates the savings potential of a job occupying a specific position in a sequence and builds a complete schedule according to these savings. Ríos-Mercado and Bard (1998a) developed a branch-and-cut algorithm which obtained optimum solutions for instances of up to eight jobs and six machines. The same authors also developed a branch-and-bound method (Ríos-Mercado and Bard, 1999a) that allowed solving instances of up to 10 jobs and six machines with a maximum deviation of 1% from the optimal solution. In Ríos-Mercado and Bard (1998b), the same authors proposed an insertion heuristic based on the NEH heuristic with the improvements of Taillard (1990), called NEHT_RMB and a metaheuristic based on a greedy randomized adaptive search procedure (GRASP). Later they proposed a TSP based heuristic based on the work of Simons which they called HYBRID (Ríos-Mercado and Bard, 1999b). Parthasarathy and Rajendran (1997) developed a simulated annealing algorithm for the SDST flowshop with job due dates and with the objective of minimising the total weighted tardiness ($\sum w_i T_i$). More recently, Tseng and Stafford (2001) have proposed two new MILP models for the SDST flowshop and managed to solve instances of up to nine jobs and nine machines in about 5 minutes of CPU time on a Pentium III running at 800 MHz. The reader can refer to Allahverdi et al. (1999) and Yang and Liao (1999) for more general and complete scheduling reviews considering setup times and to Cheng et al. (2000) for a recent review of flowshop scheduling research involving setup times.

As we have seen, some exact approaches for the SDST flowshop have been proposed, but they are only applicable to the two machine case, and those exact methods proposed for the m machine case are only practicable for very small instances. Furthermore, some general heuristics proposed, such as the lexicographic search by Gupta (1975) or the multi-sort algorithm by Szwarc and Gupta (1987), are intensive on CPU time requirements and therefore not suitable for medium or large instances. Moreover, there is almost no work regarding SDST flowshops with the $C_{\max}$ criterion and metaheuristics. The only proposed method is the cited GRASP algorithm by Ríos-Mercado and Bard (1998b).

Genetic algorithms have been proposed for the PFSP problem (see Chen et al., 1995; Reeves, 1995 or, more recently Ruiz et al., 2003) but, to the best of our knowledge, no genetic algorithms have been proposed for the SDST flowshop so far. Furthermore, no comparable evaluation of SDST flowshop sequencing techniques has been carried out, since authors usually generate their own problem

instances and the algorithms are run on different computers, making the comparison between techniques difficult and barely reproducible.

3. A genetic algorithm for the SDST flowshop

In a genetic algorithm there is a set of solutions that form the *population* and these solutions are encoded into *chromosomes*. Every individual in the population is evaluated and assigned a *fitness* value, the higher this value, the better the solution. The population *evolves* after applying a series of operations until some stopping criteria is met. A complete iteration of a genetic algorithm is called a *generation* and can be described as follows: a *selection* mechanism randomly picks from the population some members according to the assigned fitness value, in such a way that the fittest individuals have a greater chance of being selected. The selected individuals mate (in a process called *crossover*) and generate new individuals called *offspring*. After mating, some offspring might undergo a *mutation*. Afterwards, the new population is re-evaluated and the whole process is repeated (see, Goldberg, 1989; Michalewicz, 1996).

When building a GA there are several choices available: encoding, selection, population size, choice of crossover and mutation operators, crossover and mutation probability, and so on. Additionally, the effectiveness of the genetic algorithms greatly depends on the correct choice of operators and parameters. The proposed genetic algorithm has been originally applied by the authors in a previous work (see Ruiz et al., 2003) to the regular flowshop with great success. In the following sections we briefly describe the cited genetic algorithm and the specific changes for solving the SDST flowshop.

3.1. Encoding and population initialization

The most widely used encoding for the PFSP is a simple permutation of the jobs. This permutation has to be feasible, i.e., there should be no missing jobs and no job can be repeated. In a genetic algorithm the initial population is normally generated randomly. Extensive testing with the regular flowshop showed that randomly generated populations resulted in algorithms that were slow to converge. In the initialization proposed by Reeves (1995) all members are generated at random except one which is generated from the application of the NEH heuristic. This initialization results in a better algorithm. In order to better understand this initialization procedure we are going to briefly describe the NEH procedure: the NEH heuristic is based on the idea that jobs with high processing times on all the machines should be scheduled as early as possible. This heuristic can be divided into three simple steps:

1. The total processing times for the jobs on the m machines are calculated
$$\forall \text{job } i, i = 1, \ldots, n, \quad P_i = \sum_{j=1}^{m} p_{ij}.$$
2. The jobs are sorted in descending order of P_i. Then, the first two jobs (those two with higher P_i) are taken and the two possible schedules containing them are evaluated.
3. Take job i, $i = 3, \ldots, n$, and find the best schedule by placing it in all the possible i positions in the sequence of jobs that are already scheduled. For example, if $i = 4$, the previously built sequence would contain the first three jobs of the sorted list calculated in step 2, then, the fourth job could be placed either in the first, in the second, in the third or in the last position of the sequence. The best sequence of the four would be selected for the next iteration.

The proposed GA initialization procedure is based on this NEH heuristic and in the following modification: after having ordered the jobs by decreasing order of their processing times on all machines (step 2 of the NEH heuristic), we pick two random jobs from the ordered list and exchange them with the two first jobs. Then we proceed with the rest of the NEH heuristic. According to the choice of the two initial jobs considered for step 2 we will have a different final solution. By repeatedly applying this modified NEH heuristic, we will obtain several good

individuals that will form the initial population. Therefore, we propose the following initialization procedure: the first individual of the population is generated with the standard NEH heuristic, then up to $(B_i)\%$ of the initial population is filled with individuals generated from the aforementioned modified NEH heuristic. The remaining $(100 - B_i)\%$ of the population is filled with random sequences. After applying this procedure we will have a population where the first $(B_i)\%$ members are both different and good starting sequences. It is worth noting that the NEH heuristic used is based on the enhancements of Taillard (1990) and the consideration of sequence dependent setup times by Ríos-Mercado and Bard (1998b). Thus we are using the NEHT_RMB heuristic and a modified NEHT_RMB heuristic for the initialization of the population.

3.2. Selection mechanism and generational scheme

For the selection we have implemented two classical selection schemes, called roulette wheel selection and tournament selection (see Goldberg, 1989; Michalewicz, 1996). We refer to generational scheme as the process by which new individuals in a new generation replace old members from the previous generation. As was shown in Ruiz et al. (2003), a steady state GA where the offspring replace the worst individuals in the population resulted in a very effective genetic algorithm. The generational scheme was further extended by the following method; let the worst individual in a population be denoted as p_{worst} and its makespan c_{worst}. There are several additional constraints to when offspring can replace p_{worst}. A given offspring can only replace p_{worst} if its makespan is lower than c_{worst}. This approach and the high selection pressure of the two selection mechanisms implemented resulted in a premature convergence of the population. To overcome this problem we devised a new mechanism for determining when offspring can replace p_{worst}: a new individual will only replace p_{worst} if its makespan is better than c_{worst} and if its sequence is unique, i.e. the sequence is not already in the population.

It is known that there can be many different optimal sequences in the regular flowshop and the same can be said about the SDST flowshop, this is the main motivation behind the proposed generational scheme.

3.3. Crossover and mutation operators

The crossover operator creates offspring by coalescing two other sequences called parents. The aim is to generate better children, i.e. to generate better solutions after the crossover. Many different general and specific crossover operators have been proposed for the PFSP in the literature. In Ruiz et al. (2003) the authors tested several of the typical crossover operators for the PFSP (PMX or *Partially Mapped Crossover*, OP or *One Point Order Crossover* and TP or *Two Point Order Crossover*, among others) and the results showed that many times, the crossover generated offspring with worse makespan values than those of the parents. The study revealed than this outcome was due to the fact that the crossover tends to disrupt building blocks in the latter stages of the algorithm, thus creating offspring that are consistently worse than their progenitors. We proposed four new crossover operators for the PFSP that overcome this problem. These new crossover operators are based on the idea of identifying and maintaining building blocks in the crossover, considering building blocks as similar job occurrences in both parents. In this way, building blocks are passed over to offspring unaltered. If there are no similar blocks in the parents (as in the first stages of the algorithm) the proposed crossover operators behave like the one point order crossover or the two point order crossover, depending on the case.

We proposed four different crossover operators, named "Similar Job Order Crossover" or SJOX, "Similar Block Order Crossover" (SBOX), "Similar Job 2-Point Order Crossover" (SJ2OX) and "Similar Block 2-Point Order Crossover" (SB2OX). This last operator's results are superior to all other considered operators, and thus we give a detailed description here: first of all, both parents are examined on a position-by-position basis. We consider blocks of at least two consecutive identical jobs and only those identical blocks on both parents are directly copied to children (Fig. 1(a)). Then, two random cut points are taken and the

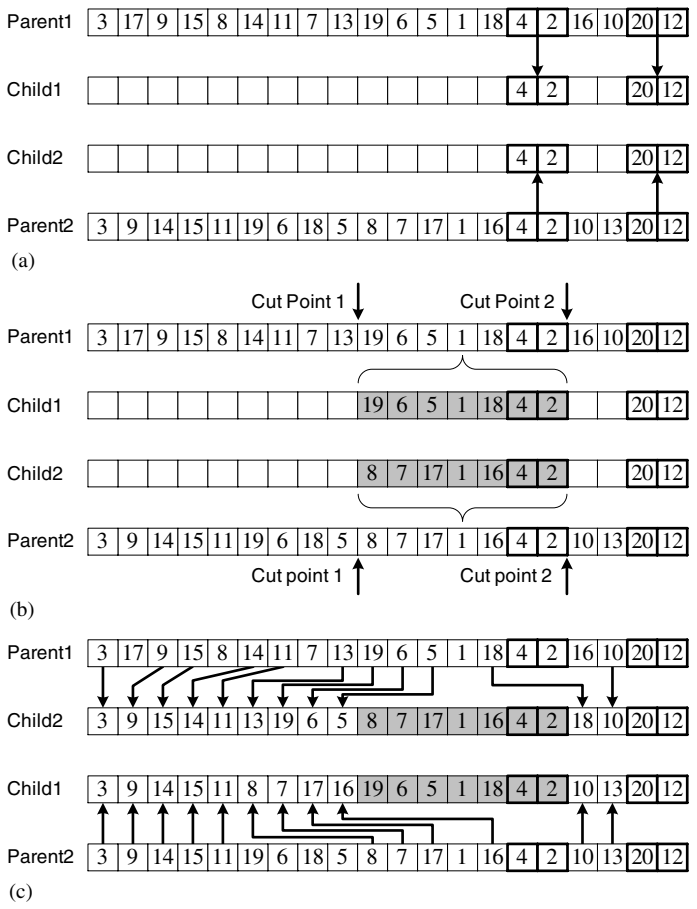

Fig. 1. "Similar Block 2-Point Order Crossover" (SB2OX): (a) first, the common blocks of jobs in both parents are copied over to the offspring; (b) then, jobs between the cut point are inherited from the direct parent and (c) the missing elements are copied in the relative order of the other parent.

section between these two points is directly copied to the children (Fig. 1(b)). Lastly, the missing elements of each offspring are copied in the relative order of the other parent (Fig. 1(c)).

A genetic algorithm includes a mutation operator to avoid convergence to local optima and to introduce lost genetic material and variability in the population. For the proposed genetic algorithm we use the SHIFT mutation, where a randomly picked position in the sequence is relocated at another randomly picked position and the jobs between these two positions move along. This mutation operator was also used by other authors in their corresponding genetic algorithms for the regular flowshop (see Reeves, 1995 and Murata et al., 1996).

3.4. Restart scheme

The same authors implemented a restart scheme to overcome the problem arising from genetic algorithms when the population achieves a low diversity and the process stalls around a local optima. We proposed a restart mechanism based on the ideas of a restart scheme used by the authors in previously related research (Alcaraz et al., 2003). This process works as follows: at each generation we store the minimum makespan. If in

the following generation the minimum makespan is the same, we increment a counter. If this counter, called *countmak*, becomes greater that a control parameter called G_r, we apply the following restart procedure:

- Sort the population in ascending order of $C_{\max}$.
- Skip the first 20% of individuals from the sorted list, i.e. the best individuals.
- From the remaining 80% individuals, 50% of them are replaced by simple SHIFT mutations of the first 20% best individuals. 25% are replaced by newly generated sequences from the modified NEHT_RMB procedure (see Section 3.1) and the remaining 25% are replaced by new randomly generated schedules. Make $countmak = 0$.

Therefore, when the lowest makespan in the population has remained unchanged for more than G_r generations, the restart procedure replaces all population members except the 20% best individuals, allowing the genetic search to restart.

3.5. Experimental evaluation of the algorithm

It is known that the different operators and levels of the parameters clearly affect the quality of the solutions obtained by a genetic algorithm. Ruiz et al. (2003) presented a thoughtful calibration of a genetic algorithm for the PFSP by means of a Design of Experiments (DOE) approach. This experimentation allowed us to obtain a finely tuned algorithm that resulted in much higher effectiveness. In this section, we study the proposed GA but applied to the SDST flowshop. A number of different genetic algorithms can be obtained with the different combinations of the factors and parameters mentioned in previous sections. We have chosen a full factorial design in which the following combinations are tested:

- *Selection type*. Two levels (Tournament and Ranking).
- *Crossover type*. Seven levels (PMX, OP, TP, SJOX, SBOX, SJ2OX and SB2OX).
- *Crossover Probability* (P_c). Six levels (0.0, 0.1, 0.2, 0.3, 0.4 and 0.5).
- *Mutation Probability* (P_m). Five levels (0.0, 0.005, 0.01, 0.015 and 0.02).
- *Population size* (P_{size}). Four levels (20, 30, 40 and 50).
- *Restart* (G_r). Three levels (25, 50 and 75).
- *Population initialization* (B_i). Fixed at 25%.

All the cited factors resulted in a total of $2 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 = 5040$ different genetic algorithms. We performed four different experiments, one for each of the four different sequence dependent Taillard-based instance sets from Vallada et al. (2003). The tests contain four different processing times to sequence dependent setup times ratios. i.e., the instance set SSD-10 is composed of 120 instances where the p_{ij} are those of Taillard's benchmark and where the sequence dependent setup times are 10% of the processing times. In the instance set SSD-50, the setup times are 50% of the processing times and the instance sets SSD-100 and SSD-125 have setup times that are 100% and 125% of the processing times respectively. So for example, if the p_{ij} in Taillard's instances are generated from a uniform distribution in the range $[1, 99]$, in the SSD-10 instance set the setup times are uniformly distributed in the range $[1, 9]$ ($[1, 49]$, $[1, 99]$ and $[1, 124]$ for the instance sets SSD-50, SSD-100 and SSD-125 respectively). Thus, we have four problem sets and a total of 480 different instances. These benchmarks are available from http://www.upv.es/gio. The stopping criterion was set at 5000 makespan evaluations for all different genetic algorithms. The 200 and 500 job instances were rather large and we chose to solve only the first 90 instances (up to 100 jobs and 20 machines). The response variable is based on the following performance measure:

$$\%\ \text{Increase over the best solution} = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \times 100, \quad (1)$$

where Heu_{sol} is the best makespan obtained by a given algorithm and $Best_{sol}$ is the makespan of the best solution obtained after carrying out several runs where 100 000 makespans are evaluated by an iterated local search algorithm. This iterated local search is based on the insertion neighbourhood

and starts from the solution given by the cited NEHT_RMB heuristic. The response variable is, therefore, the average percentage increase over the lowest known upper bound. In total we have $5040 \cdot 4 \cdot 90 = 1\,814\,400$ different runs. All tests were performed in a cluster of four PC/AT computers with Athlon XP 1600+ processors and 512 MBytes of main memory.

The resulting full factorial experiment was analysed by means of a multi-factor Analysis of Variance (ANOVA) technique. The resulting ANOVA table for the SSD-10 instance set experiment is shown in Table 1.

For analysing the experiment we will focus on the F-ratio, which is a ratio between the variance explained by a factor and the unexplained variance. It is worth noting that all interactions between more than two factors have not been studied as their F-ratios were negligible. The first factor or interaction that is analysed is the one corresponding to the greatest F-ratio. To set the level of the factor we used a means plot to graphically see which level was best for the genetic algorithm. Then we followed the same procedure for the second greatest F-ratio, then for the third and so on. This procedure was repeated until all experimental factors were fixed to some level.

As seen in the ANOVA table, the greatest F-ratio corresponds to "Select_Type" or the type of selection. This factor has two levels, that is, there are two different selection methods. The two means are plotted in Fig. 2.

From the plot we can see that there is a statistically significant difference between Tournament and Ranking selection schemes. Clearly, Tournament selection ($\simeq$1.39% average increase) results in a much more effective genetic algorithm than Ranking selection ($\simeq$1.52% average increase). The

Table 1
ANOVA table for the genetic algorithm experiment. SSD-10 instance set

Source	Sum of squares	Df	Mean square	F-ratio	P-value
Analysis of variance for medias—type III sums of squares					
Main effects					
A: Cross_Type	1.21849	6	0.203082	145.35	0.0000
B: Pop_Size	0.383839	3	0.127946	91.57	0.0000
C: Select_Type	21.5889	1	21.5889	15451.61	0.0000
D: Cross_Prob	1.81791	5	0.363583	260.22	0.0000
E: Mut_Prob	11.0408	4	2.76021	1975.54	0.0000
F: Restart	6.94995	2	3.47497	2487.11	0.0000
Interactions					
AB	0.0811206	18	0.0045067	3.23	0.0000
AC	0.0790907	6	0.0131818	9.43	0.0000
AD	0.348617	30	0.0116206	8.32	0.0000
AE	0.521944	24	0.0217477	15.57	0.0000
AF	0.0318703	12	0.00265586	1.90	0.0297
BC	1.22839	3	0.409462	293.06	0.0000
BD	0.207063	15	0.0138042	9.88	0.0000
BE	0.170607	12	0.0142172	10.18	0.0000
BF	0.0090561	6	0.00150935	1.08	0.3716
CD	0.592647	5	0.118529	84.83	0.0000
CE	4.38832	4	1.09708	785.20	0.0000
CF	0.0109267	2	0.00546336	3.91	0.0201
DE	10.2519	20	0.512594	366.87	0.0000
DF	0.267935	10	0.0267935	19.18	0.0000
EF	0.55091	8	0.0688637	49.29	0.0000
Residual	6.7666	4843	0.00139719		
Total (corrected)	68.5068	5039			

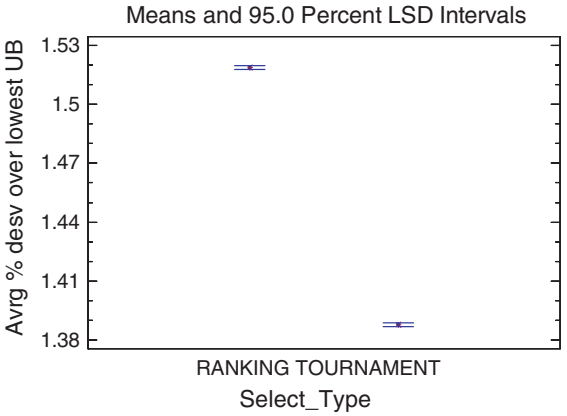

Fig. 2. Means plot for the type of selection factor. SSD-10 instance set.

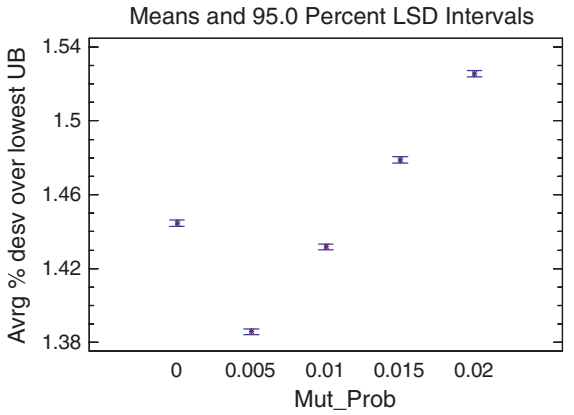

Fig. 4. Means plot for the probability of mutation (P_m) factor. SSD-10 instance set.

next greatest F-ratio corresponds to the Restart factor (G_r), in Fig. 3 we can see the three plotted levels.

From the figure it is clear than the level $G_r = 25$, results in a better genetic algorithm. The third considered factor is the mutation rate or P_m. Fig. 4 shows the five different mutation probabilities.

We can see that as the mutation rate decreases the genetic algorithm's effectiveness increases until we reach the best mutation rate, which is 0.005 and then, if we further decrease the mutation rate (0.0) the resulting GA is clearly worse. Therefore, the proposed GA works well with mutation, albeit at a low rate.

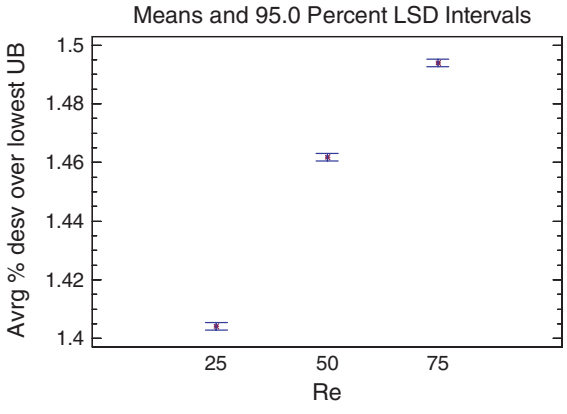

Fig. 3. Means plot for Restart (G_r) factor. SSD-10 instance set.

Interactions between factors have to be studied as well. The fourth greatest F-ratio corresponds to the interaction between the factors "Select_Type" and "Mut_Prob", that is between the selection type and the mutation rate or probability. Fig. 5 shows the corresponding means plot.

We see that the mutation rate is not affected by the selection type until we reach a mutation rate of 0.0. This plot confirms previous findings, the best combination being a mutation rate of 0.005 and a tournament selection scheme. Lastly, we analyse the performance of the different tested crossover operators. It is well known that the crossover operator affects the performance of a GA the most (see Alcaraz and Maroto, 2001). The corresponding means plot can be seen in Fig. 6.

There are statistically significant differences between the seven considered crossover operators. Two of our proposed crossover operators, the SB2OX and the SJ2OX are significantly better than the others, and all four proposed crossover operators outperform the classical PMX operator which has been regarded as the best crossover operator for the PFSP by some authors (see Chen et al., 1995 and Goldberg, 1989). The SB2OX operator seems to be the best crossover operator among all seven.

For brevity, we have omitted the remaining analysis, but after having completed the calibration of the GA we obtained the following configuration of the final genetic algorithm: Selection

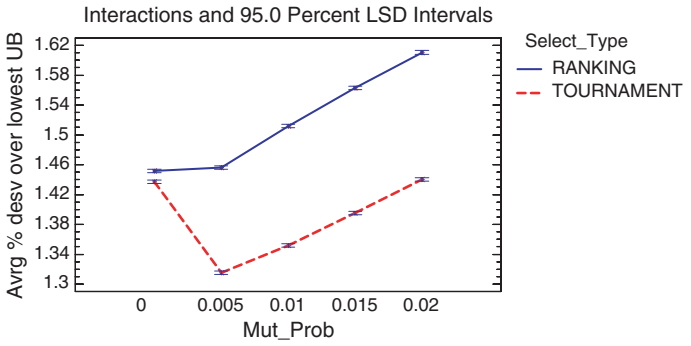

Fig. 5. Means plot for the interaction between the selection type and the probability of mutation factors. SSD-10 instance set.

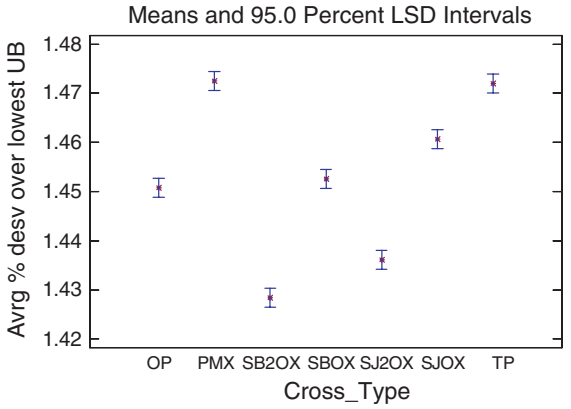

Fig. 6. Means plot for the crossover operator factor.

type: Tournament, Crossover type: Similar Block 2-Point Order Crossover (SB2OX), Crossover Probability (P_c): 0.1, Mutation Type: SHIFT, Mutation Probability (P_m): 0.005, Population size (P_{size}): 50 and finally, Restart (G_r): 25.

The ANOVA analysis for the SSD-50, SSD-100 and SSD-125 instance sets yielded identical results. This is a remarkable outcome, since we can safely say that the proposed GA is robust regardless of the ratio between the setup times and processing times.

4. Hybridization of the genetic algorithm

After the calibration of the proposed genetic algorithm we aimed to obtain better results by means of a local search. We propose an adaptation of the proposed hybrid genetic algorithm for the SDST flowshop. This hybrid genetic algorithm is now summarized.

Murata et al. (1996) presented an improvement phase by applying a local search step to all members of the population at each generation in a genetic algorithm. The problem of this approach is that it results in a slow genetic algorithm. In Ruiz et al. (2003), we showed the application of an improvement phase after selection, crossover and mutation, but not to every individual in the population. We define an "enhancement probability" or P_{enh} where the individual undergoes an enhancement according to this rate. In this way we can finely tune the level of improvement versus the computer processing requirements depending on the value of P_{enh}. For the SDST flowshop we propose the same local search as Ríos-Mercado and Bard (1998b), which is based on the insertion neighborhood and in the `NEH_RMB` heuristic. This heuristic includes the acceleration of the NEH method proposed by Taillard (1990).

From this form of local search, we obtain an enhanced genetic algorithm resulting from the hybridization of the GA detailed in Section 3. After selection, crossover and mutation, we improve a given individual according to P_{enh} a given number Enh_s steps or iterations of the cited local search procedure. The individual obtained after enhancement is accepted only if improved. The parameters P_{enh} and Enh_s can be adjusted to match the computing time requirements. The higher these two parameters, the slower the resulting algorithm.

For low values of P_{enh} the hybridization had little impact on the GA's effectiveness and a clear impact on the computing time needed. Therefore, instead of raising P_{enh} (and thus making the algorithm more effective but slower) we showed that, after each generation, the best individual of the population should be enhanced by a rate of $2 \cdot P_{enh}$. The problem is that the best individual can be a strong local optima and the hybrid GA might spend many local search iterations unsuccessfully trying to improve it. We devised a "best individual improvement" strategy that is explained as follows: for every individual in the population, we annotate the number of times it has been enhanced without success. We initialize a parameter called *mult* to 10. If the best individual of the population has not improved after more than $Enh_s \cdot mult$ steps, then we try to improve the second best individual. If the situation for the second individual is the same we look for the third and so on. There can be a case where, in a given generation, all individuals in the population have not improved for more than $Enh_s \cdot mult$ steps. In this case, we make $mult = mult * 2$ and repeat the process again, thus allowing for more local search steps for every individual. With this strategy, only the best individual that has been unsuccessfully improved for less than $Enh_s \cdot mult$ local search iterations will undergo enhancement with a rate of $2 \cdot P_{enh}$ at each generation. This procedure avoids the problem of spending too much time trying to improve a strong local optima and therefore, invalidating the improvement phase and the hybrid algorithm altogether. More details on this hybrid genetic algorithm can be found in Ruiz et al. (2003).

5. Adaptation of existing metaheuristics

We will now describe in detail the metaheuristics that we have adapted for application to the SDST flowshop. These methods include known metaheuristics originally applied to the regular flowshop ($F//C_{\max}$) that have shown to have a superior performance in comparison with other methods tested (see Ruiz and Maroto, 2004). The methods that we adapt here have not been applied to the SDST flowshop before, and we expect them to yield good results.

First of all, we have adapted the genetic algorithm of Reeves (1995) which we will call `GA_REEV`. This algorithm includes an initialization that uses the NEH algorithm and therefore we have changed this initialization to use the NEHT_RMB method of Ríos-Mercado and Bard (1998b). Also, the evaluation of the individuals at each generation has been modified so as to take into account the existence of sequence dependent setup times. Another adapted metaheuristic is the simulated annealing of Osman and Potts (1989), which we will refer to as `SA_OP`. In this case, only the evaluation of the incumbent solution at each generation has been modified to take into account the setup times. A very effective metaheuristic is the iterated local search procedure of Stützle (1998), which we will call `ILS`. This metaheuristic has an NEH initialization like the `GA_REEV` algorithm and therefore, we have also adapted this method for the SDST flowshop by using the NEHT_RMB heuristic for the initialization and by considering setup times when evaluating the incumbent solution at each iteration. Another modified metaheuristic is the tabu search of Widmer and Hertz (1989) (`SPIRIT`). For this last method we have chosen an `NEHT_RMB` initialization, since the initialization proposed by Widmer and Hertz (based on a TSP representation of the regular flowshop problem) does not take into account sequence dependent setup times. Lastly, we have also adapted a recent genetic algorithm applied to the regular flowshop problem with no-wait constraints in Aldowaisan and Allahverdi (2003), which we will refer to as `GA_AA`. As in all other cases, the changes performed to this algorithm involve the initialization where the no-wait specific heuristics have been substituted by the `NEHT_RMB`. The evaluation of the individuals at each generation of the algorithm has also been changed to take into account the setup times.

6. Computational experience

In this section we are going to compare the two genetic algorithms proposed (referred to as `GA` and

HGA), with the adapted metaheuristics proposed in the last section and with other published specific methods for the SDST flowshop. For the hybrid GA (HGA) we have fixed the local search parameters to $P_{enh} = 0.05$ and $Enh_s = 25$.

The SDST specific methods compared are: the NEH heuristic by Nawaz et al. (1983) with the enhancements of Taillard (1990) and the consideration of sequence dependent setup times by Ríos-Mercado and Bard (1998b) that will be denoted as NEHT_RMB; the greedy randomized adaptive search procedure of Ríos-Mercado and Bard (1998b), referred to as RMB_GRASP; the Total and Setup heuristics by Simons (1992) (S_TOTAL and S_SETUP); the TSP-based heuristic of Ríos-Mercado and Bard (1999b), denoted as RMB_HYB and the savings index algorithm of Das et al. (1995), which will be referred to as SI_DGK. We have also included a simple method that generates and evaluates a number of random schedules and returns the best one as a result. This method will be referred to as RANDOM.

For the results to be comparable we have fixed a maximum of 10 000 makespan evaluations for all the methods tested that rely on a stopping criteria (i.e. GA, HGA, RMB_GRASP, GA_REEV, SA_OP, ILS, SPIRIT, GA_AA and RANDOM). For evaluating the different methods (14 in total) we used the performance measure stated in expression (1). A total of five independent runs were performed for the stochastic methods (GA, HGA, RMB_GRASP, GA_REEV, SA_OP, ILS, GA_AA and RANDOM) and the results were finally averaged. For the comparison we have used a PC/AT computer with an Athlon XP 1600+ processor and 512 MBytes of main memory. Table 2 shows the results for the SSD-10 and SSD-50 instance sets and Table 3 for the SSD-100 ad SSD-125 instance sets. The results are grouped by instance type and size and at each cell we have the average percentage increase and the CPU times needed in parenthesis (times in seconds).

It is clear from the computational results that the HGA algorithm exceeds all other methods by a wide margin. From the SSD-10 instance set we can see that the two best algorithms are the HGA and the GA with average percentage increases of 0.33%, and 1.04%, respectively. The third best method is the adapted ILS algorithm with a 1.61% average percentage increase. This means that the HGA method is a sound 488% better than the ILS, albeit being almost 33% slower on average. However, the GA method is a noteworthy 55% better than the ILS and both methods are comparable in terms of speed. Most remarkably, the SDST flowshop specific heuristics, S_TOTAL, S_SETUP and RMB_HYB are very fast but show poor results, especially if compared with NEHT_RMB, which obtains solutions even faster and with a much lower average percentage increase. Furthermore, it can be said that the simplistic RANDOM rule is more effective than these three methods although it is slower. Another salient conclusion from the computational study is the RMB_GRASP metaheuristic, which obtains fairly good results (1.69% average percentage increase) although the CPU times required do not scale well with the instance size, for the 200×20 instances, this method needed more than 30 minutes on average to obtain a solution. As a matter of fact, no solution could be obtained in less than 3 hours for the 500×20 instances (shown as (*) in Tables 2 and 3). Of all the SDST flowshop specific methods, the SI_DGK is the one that shows the worst results. Its average percentage deviation is slightly lower than the S_TOTAL and S_SETUP heuristics, but the CPU times needed are the highest in the comparative, needing almost 50 minutes of running time for the 200×20 instances. The estimated running time for the 500×20 instances was more than 40 hours and thus, no result was obtained for this instance size (shown as (*) in Tables 2 and 3).

For the SSD-50 instances, we again find that the two best algorithms are the HGA and the GA. In this case, there is an astounding difference of 385% between the HGA and the third best method, which in this case is the RMB_GRASP metaheuristic.

For the SSD-100 and SSD-125 instance sets we obtain the same pattern, the two best methods are still the HGA and the GA and the third best is the RMB_GRASP metaheuristic. In these cases the differences are 366% for the SSD-100 case and 368% for the SSD-125 instance set. What we can observe is that as the ratio between the setup times and processing times increases the algorithms appear to obtain slightly worse average percentage

Table 2
Average percentage increase over the best known upper bound and elapsed CPU times in seconds for the methods evaluated with the termination criteria set at 10 000 makespan evaluations. Instances SSD-10 and SSD-50

Instance	RANDOM	NEHT_RMB	GA	ILS	SA_OP	SPIRIT	GA_REEV	HGA	S_TOTAL	S_SETUP	RMB_HYB	RMB_GRASP	SI_DGK	GA_AA
SSD-10 instances														
20×5	8.45	3.77	0.41	0.76	2.08	2.03	1.23	**0.09**	25.10	23.00	15.62	1.06	22.14	1.70
	(<0.5)	(<0.5)	(<0.5)	(0.54)	(<0.5)	(<0.5)	(<0.5)	(2.68)	(<0.5)	(<0.5)	(<0.5)	(2.68)	(<0.5)	(<0.5)
20×10	10.09	4.31	0.52	1.28	2.79	2.73	1.90	**0.18**	24.35	27.89	18.25	1.33	21.44	2.52
	(<0.5)	(<0.5)	(0.51)	(0.93)	(<0.5)	(<0.5)	(<0.5)	(2.66)	(<0.5)	(<0.5)	(<0.5)	(3.83)	(<0.5)	(<0.5)
20×20	8.64	3.55	0.54	1.31	2.85	2.68	1.65	**0.17**	19.35	19.35	14.29	1.25	17.37	2.42
	(<0.5)	(<0.5)	(0.71)	(1.21)	(<0.5)	(<0.5)	(<0.5)	(3.43)	(<0.5)	(<0.5)	(<0.5)	(6.49)	(<0.5)	(<0.5)
50×5	9.32	2.84	0.83	1.44	2.21	1.74	1.38	**0.23**	20.24	18.22	14.54	1.73	21.33	2.55
	(<0.5)	(<0.5)	(0.61)	(1.09)	(<0.5)	(<0.5)	(<0.5)	(3.96)	(<0.5)	(<0.5)	(<0.5)	(15.49)	(1.01)	(<0.5)
50×10	13.58	4.14	1.29	2.22	3.75	2.99	2.28	**0.45**	23.85	23.34	17.95	2.22	23.01	3.82
	(<0.5)	(<0.5)	(0.94)	(2.31)	(<0.5)	(<0.5)	(0.59)	(4.82)	(<0.5)	(<0.5)	(<0.5)	(24.27)	(2.02)	(<0.5)
50×20	14.32	3.83	1.29	2.40	3.27	2.92	2.56	**0.33**	22.44	23.99	19.31	2.20	21.37	3.68
	(0.58)	(<0.5)	(1.75)	(3.95)	(0.61)	(0.58)	(0.90)	(8.71)	(<0.5)	(<0.5)	(<0.5)	(45.84)	(4.12)	(0.92)
100×5	8.89	2.47	1.25	1.64	2.21	2.19	1.69	**0.29**	17.00	16.92	12.29	1.75	19.43	2.43
	(<0.5)	(<0.5)	(1.02)	(2.79)	(<0.5)	(<0.5)	(0.97)	(6.34)	(<0.5)	(<0.5)	(<0.5)	(65.64)	(16.30)	(0.71)
100×10	12.50	2.94	1.28	1.87	3.45	2.56	2.04	**0.32**	20.57	20.63	16.74	1.81	20.74	2.90
	(1.25)	(<0.5)	(2.00)	(4.78)	(0.63)	(0.60)	(1.31)	(11.49)	(<0.5)	(<0.5)	(<0.5)	(119.32)	(33.02)	(1.44)
100×20	14.38	3.13	1.58	2.29	4.26	2.85	2.35	**0.51**	21.06	21.48	17.96	2.14	18.11	3.12
	(3.95)	(<0.5)	(5.13)	(13.23)	(1.35)	(1.32)	(2.13)	(27.21)	(<0.5)	(<0.5)	(<0.5)	(269.52)	(68.84)	(3.75)
200×10	11.19	2.04	1.37	1.62	3.43	1.94	1.66	**0.50**	18.49	18.19	13.82	1.50	17.35	2.03
	(4.85)	(<0.5)	(6.44)	(16.74)	(1.49)	(1.57)	(3.87)	(28.61)	(<0.5)	(<0.5)	(1.73)	(706.63)	(603.74)	(5.22)
200×20	13.65	2.28	1.39	1.67	4.52	2.16	1.90	**0.50**	18.21	19.21	16.52	1.60	17.38	2.27
	(11.82)	(<0.5)	(25.35)	(39.57)	(10.68)	(11.03)	(13.49)	(61.54)	(<0.5)	(<0.5)	(2.16)	(1917.72)	(2994.50)	(13.84)
500×20	10.86	1.09	0.77	0.85	4.21	1.06	0.95	**0.37**	14.57	14.77	12.77	(*)	(*)	1.09
	(38.75)	(1.53)	(141.21)	(114.59)	(31.13)	(32.62)	(45.36)	(177.93)	(2.49)	(2.50)	(41.70)			(41.90)
Average	11.32	3.03	1.04	1.61	3.25	2.32	1.80	**0.33**	20.43	20.58	15.84	1.69	19.97	2.68
	(5.19)	(<0.5)	(15.51)	(16.81)	(3.92)	(4.06)	(5.83)	(28.28)	(<0.5)	(<0.5)	(3.88)	(288.86)	(338.52)	(2.49)

SSD-50 instances

20×5	13.96	6.53	1.00	2.50	4.55	4.98	3.32	**0.50**	27.68	25.57	20.75	2.81	18.38	4.84
	(<0.5)	(<0.5)	(<0.5)	(0.66)	(<0.5)	(<0.5)	(<0.5)	(2.37)	(<0.5)	(<0.5)	(<0.5)	(2.67)	(<0.5)	(<0.5)
20×10	11.94	6.30	1.09	2.17	4.68	4.67	2.93	**0.32**	27.00	24.88	18.35	2.31	18.29	4.09
	(<0.5)	(<0.5)	(<0.5)	(0.83)	(<0.5)	(<0.5)	(<0.5)	(2.50)	(<0.5)	(<0.5)	(<0.5)	(3.93)	(<0.5)	(<0.5)
20×20	9.06	4.09	0.42	1.66	3.02	3.62	2.06	**0.18**	19.09	19.83	15.39	1.45	15.34	2.98
	(<0.5)	(<0.5)	(0.67)	(1.28)	(<0.5)	(<0.5)	(<0.5)	(3.42)	(<0.5)	(<0.5)	(<0.5)	(6.34)	(<0.5)	(<0.5)
50×5	20.05	6.77	3.03	5.40	5.93	5.73	4.56	**0.92**	31.47	32.28	25.29	4.61	16.22	6.58
	(<0.5)	(<0.5)	(0.61)	(1.23)	(<0.5)	(<0.5)	(<0.5)	(3.50)	(<0.5)	(<0.5)	(<0.5)	(15.52)	(1.01)	(<0.5)
50×10	17.77	6.14	2.77	4.71	5.53	5.33	4.18	**1.13**	27.77	27.88	23.74	4.22	17.74	5.95
	(<0.5)	(<0.5)	(0.92)	(2.05)	(<0.5)	(<0.5)	(0.58)	(4.63)	(<0.5)	(<0.5)	(<0.5)	(24.18)	(2.00)	(<0.5)
50×20	15.22	4.92	1.97	4.12	4.38	4.58	3.86	**0.61**	25.45	23.19	19.87	3.16	17.46	4.79
	(0.60)	(<0.5)	(1.71)	(3.59)	(0.61)	(0.57)	(0.89)	(10.08)	(<0.5)	(<0.5)	(<0.5)	(46.06)	(4.13)	(0.74)
100×5	21.39	5.76	3.81	5.53	7.16	5.37	4.92	**1.28**	29.39	30.66	25.36	4.63	13.70	5.75
	(<0.5)	(<0.5)	(1.17)	(3.09)	(<0.5)	(<0.5)	(0.97)	(6.33)	(<0.5)	(<0.5)	(<0.5)	(66.78)	(16.21)	(0.55)
100×10	18.65	5.13	3.14	4.81	6.67	4.76	4.30	**1.18**	25.90	27.15	23.02	3.90	15.63	5.11
	(1.20)	(<0.5)	(2.32)	(5.47)	(0.62)	(0.60)	(1.31)	(11.40)	(<0.5)	(<0.5)	(<0.5)	(120.12)	(32.96)	(1.14)
100×20	16.05	4.21	2.47	3.99	5.75	3.96	3.58	**0.89**	23.07	21.81	19.15	3.12	13.73	4.21
	(3.74)	(<0.5)	(5.94)	(12.90)	(1.35)	(1.31)	(2.11)	(27.53)	(<0.5)	(<0.5)	(<0.5)	(268.32)	(68.66)	(3.03)
200×10	17.89	3.66	2.71	3.59	7.13	3.58	3.38	**1.29**	23.76	24.02	20.88	3.17	11.00	3.66
	(4.64)	(<0.5)	(9.16)	(15.84)	(1.46)	(1.57)	(3.79)	(27.83)	(<0.5)	(<0.5)	(1.73)	(685.40)	(602.84)	(4.16)
200×20	15.84	3.25	2.17	3.18	6.60	3.19	3.00	**1.15**	20.54	20.29	17.92	2.52	11.05	3.25
	(11.38)	(<0.5)	(29.49)	(37.91)	(10.11)	(10.48)	(12.95)	(60.42)	(<0.5)	(<0.5)	(2.22)	(1657.36)	(3018.58)	(11.63)
500×20	13.59	1.52	0.96	1.52	6.50	1.51	1.47	**0.75**	17.01	17.61	15.15	(*)	(*)	1.52
	(37.65)	(1.48)	(167.06)	(110.23)	(29.83)	(31.39)	(44.01)	(193.20)	(2.35)	(2.34)	(39.99)			(35.36)
Average	15.95	4.86	2.13	3.60	5.66	4.27	3.46	**0.85**	24.85	24.60	20.41	3.26	15.32	4.66
	(5.02)	(<0.5)	(18.33)	(16.26)	(3.76)	(3.91)	(5.66)	(29.44)	(<0.5)	(<0.5)	(3.74)	(263.33)	(340.60)	(2.04)

Table 3
Average percentage increase over the best known upper bound and elapsed CPU times in seconds for the methods evaluated with the termination criteria set at 10 000 makespan evaluations. Instances SSD-100 and SSD-125

Instance	RANDOM	NEHT_RMB	GA	ILS	SA_OP	SPIRIT	GA_REEV	HGA	S_TOTAL	S_SETUP	RMB_HYB	RMB_GRASP	SI_DGK	GA_AA
SSD-100 instances														
20×5	18.63	10.18	2.04	5.00	7.42	8.52	5.78	**0.60**	36.76	34.86	27.97	4.04	14.21	8.17
	(<0.5)	(<0.5)	(<0.5)	(<0.5)	(<0.5)	(<0.5)	(<0.5)	(2.23)	(<0.5)	(<0.5)	(<0.5)	(2.89)	(<0.5)	(<0.5)
20×10	14.26	7.23	1.66	3.74	6.17	5.75	4.54	**0.43**	28.50	29.05	22.34	3.04	16.28	5.75
	(<0.5)	(<0.5)	(<0.5)	(0.64)	(<0.5)	(<0.5)	(<0.5)	(2.61)	(<0.5)	(<0.5)	(<0.5)	(4.34)	(<0.5)	(<0.5)
20×20	10.22	5.32	1.01	2.47	4.02	4.28	3.10	**0.27**	21.67	21.80	16.20	2.14	13.97	4.02
	(<0.5)	(<0.5)	(0.68)	(1.03)	(<0.5)	(<0.5)	(<0.5)	(3.55)	(<0.5)	(<0.5)	(<0.5)	(7.22)	(<0.5)	(<0.5)
50×5	29.06	10.00	4.46	8.89	9.26	8.57	7.33	**1.64**	43.50	43.50	36.40	6.79	12.60	9.74
	(<0.5)	(<0.5)	(0.61)	(1.27)	(<0.5)	(<0.5)	(<0.5)	(3.86)	(<0.5)	(<0.5)	(<0.5)	(17.59)	(1.01)	(<0.5)
50×10	22.35	7.33	3.63	6.65	7.42	6.29	5.85	**1.49**	34.38	34.04	28.38	5.49	14.49	7.20
	(<0.5)	(<0.5)	(0.93)	(1.77)	(<0.5)	(<0.5)	(0.58)	(4.68)	(<0.5)	(<0.5)	(<0.5)	(27.09)	(2.01)	(<0.5)
50×20	16.73	5.46	2.63	4.88	5.58	4.88	4.46	**0.77**	25.45	25.32	21.37	3.77	14.93	5.40
	(0.59)	(<0.5)	(1.75)	(3.45)	(0.62)	(0.57)	(0.90)	(8.61)	(<0.5)	(<0.5)	(<0.5)	(52.23)	(4.13)	(0.74)
100×5	32.57	8.56	5.56	8.38	11.34	8.16	7.71	**1.91**	43.14	42.87	37.41	6.97	8.02	8.56
	(<0.5)	(<0.5)	(1.17)	(2.86)	(<0.5)	(<0.5)	(0.97)	(5.25)	(<0.5)	(<0.5)	(<0.5)	(75.45)	(16.32)	(0.55)
100×10	23.65	6.22	3.96	6.12	9.06	6.01	5.44	**1.54**	33.27	32.51	27.93	4.97	11.38	6.22
	(1.20)	(<0.5)	(2.46)	(5.06)	(0.62)	(0.60)	(1.30)	(11.37)	(<0.5)	(<0.5)	(<0.5)	(137.38)	(33.07)	(1.13)
100×20	18.56	5.50	3.23	5.38	7.53	5.30	4.97	**1.43**	25.65	24.29	22.02	4.03	11.09	5.50
	(3.75)	(<0.5)	(6.13)	(12.61)	(1.35)	(1.31)	(2.11)	(26.17)	(<0.5)	(<0.5)	(<0.5)	(316.33)	(68.72)	(3.01)
200×10	24.24	5.13	3.71	5.12	10.76	5.09	4.90	**2.03**	31.43	30.99	27.57	4.41	6.68	5.13
	(4.63)	(<0.5)	(9.65)	(16.53)	(1.50)	(1.63)	(3.79)	(28.45)	(<0.5)	(<0.5)	(1.74)	(822.37)	(602.12)	(4.07)
200×20	18.61	3.44	2.56	3.43	8.48	3.42	3.32	**1.55**	24.23	24.05	21.02	3.16	7.80	3.44
	(11.41)	(<0.5)	(30.55)	(38.21)	(10.00)	(10.48)	(12.97)	(63.26)	(<0.5)	(<0.5)	(2.19)	(2064.65)	(3130.76)	(11.59)
500×20	16.83	1.71	1.05	1.71	8.75	1.70	1.68	**0.91**	20.20	20.59	18.70	(*)	(*)	1.71
	(37.56)	(1.47)	(168.33)	(110.05)	(29.79)	(31.22)	(43.85)	(200.33)	(2.33)	(2.32)	(39.59)			(35.40)
Average	20.48	6.34	2.96	5.15	7.98	5.66	4.92	**1.21**	30.68	30.32	25.61	4.44	11.95	6.28
	(5.01)	(<0.5)	(18.60)	(16.16)	(3.75)	(3.90)	(5.65)	(30.03)	(<0.5)	(<0.5)	(3.71)	(320.68)	(350.76)	(2.02)

SSD-125 instances

20×5	20.87	10.61	2.20	5.02	8.39	9.30	6.06	**0.91**	40.57	40.96	30.30	4.68	13.32	8.56
	(<0.5)	(<0.5)	(<0.5)	(0.53)	(<0.5)	(<0.5)	(<0.5)	(2.27)	(<0.5)	(<0.5)	(<0.5)	(2.49)	(<0.5)	(<0.5)
20×10	15.06	7.31	1.69	3.77	6.23	5.54	4.34	**0.39**	29.53	29.37	22.88	3.19	15.10	5.28
	(<0.5)	(<0.5)	(0.52)	(0.83)	(<0.5)	(<0.5)	(<0.5)	(2.42)	(<0.5)	(<0.5)	(<0.5)	(3.91)	(<0.5)	(<0.5)
20×20	10.45	5.23	1.13	2.34	4.52	3.79	3.31	**0.27**	21.94	20.52	15.81	2.30	12.72	3.57
	(<0.5)	(<0.5)	(0.71)	(1.02)	(<0.5)	(<0.5)	(<0.5)	(3.50)	(<0.5)	(<0.5)	(<0.5)	(6.43)	(<0.5)	(<0.5)
50×5	33.30	11.31	5.62	10.12	10.73	9.93	8.39	**2.17**	48.47	49.03	40.46	8.09	11.65	11.05
	(<0.5)	(<0.5)	(0.64)	(1.04)	(<0.5)	(<0.5)	(<0.5)	(3.47)	(<0.5)	(<0.5)	(<0.5)	(15.77)	(1.05)	(<0.5)
50×10	23.91	8.19	4.11	7.52	8.33	7.64	6.90	**1.35**	36.29	35.81	29.42	6.01	13.83	8.05
	(<0.5)	(<0.5)	(0.98)	(1.72)	(<0.5)	(<0.5)	(0.57)	(4.66)	(<0.5)	(<0.5)	(<0.5)	(24.35)	(2.01)	(0.50)
50×20	18.03	6.88	3.24	5.87	6.26	5.60	5.18	**1.15**	27.00	26.56	23.13	4.60	14.02	6.62
	(0.58)	(<0.5)	(1.86)	(3.26)	(0.60)	(0.57)	(0.89)	(8.71)	(<0.5)	(<0.5)	(<0.5)	(46.15)	(4.12)	(0.93)
100×5	36.33	9.54	5.95	9.35	12.51	9.12	8.52	**2.19**	48.64	48.95	42.19	7.61	6.80	9.52
	(<0.5)	(<0.5)	(1.25)	(2.51)	(<0.5)	(<0.5)	(0.96)	(5.36)	(<0.5)	(<0.5)	(<0.5)	(67.18)	(16.38)	(0.71)
100×10	25.93	6.86	4.30	6.64	9.98	6.38	5.97	**1.60**	35.03	36.05	30.17	5.57	10.27	6.85
	(1.21)	(<0.5)	(2.52)	(4.95)	(0.62)	(0.59)	(1.30)	(10.14)	(<0.5)	(<0.5)	(<0.5)	(119.95)	(33.12)	(1.40)
100×20	19.44	5.43	3.38	5.28	7.67	5.08	4.85	**1.36**	25.74	26.26	22.52	4.09	9.94	5.42
	(3.86)	(<0.5)	(6.64)	(13.07)	(1.35)	(1.31)	(2.11)	(27.21)	(<0.5)	(<0.5)	(<0.5)	(266.98)	(68.81)	(3.54)
200×10	26.86	5.49	4.00	5.48	11.96	5.45	5.26	**2.30**	34.27	34.18	30.03	4.83	5.75	5.49
	(4.69)	(<0.5)	(10.39)	(16.94)	(1.44)	(1.56)	(3.80)	(27.75)	(<0.5)	(<0.5)	(1.71)	(687.60)	(606.56)	(4.89)
200×20	19.63	4.08	2.71	4.08	8.96	4.00	3.85	**1.54**	23.99	24.69	22.69	3.29	6.65	4.08
	(11.25)	(<0.5)	(30.90)	(38.35)	(9.85)	(10.32)	(12.76)	(62.28)	(<0.5)	(<0.5)	(2.16)	(1701.24)	(3063.92)	(13.76)
500×20	18.09	1.82	1.14	1.82	9.54	1.81	1.79	**0.95**	21.48	21.75	20.08	(*)	(*)	1.82
	(37.29)	(1.47)	(168.19)	(109.81)	(29.50)	(31.03)	(43.59)	(200.58)	(2.34)	(2.33)	(39.86)			(42.00)
Average	22.33	6.90	3.29	5.61	8.76	6.14	5.37	**1.35**	32.75	32.84	27.47	4.93	10.91	6.77
	(4.99)	(<0.5)	(18.75)	(16.17)	(3.71)	(3.87)	(5.61)	(29.86)	(<0.5)	(<0.5)	(3.73)	(267.46)	(345.11)	(2.44)

deviations, for example the HGA method obtained an average deviation of 0.33% for the SSD-10 set and for the SSD-125 set the average percentage increase increased to 1.35%. The only exception is this trend is the SI_DGK heuristic. This heuristic is specially designed to obtain better solutions when the setup times are significantly greater than the processing times. In this case we can see that for the SSD-10 instance set, its average percentage deviation is 19.97%, and for the SSD-50 15.32%. So, a 100% increase in the setup times to processing times ratio has yielded a 30% performance increase for this heuristic. However this trend softens in the SSD-100 instance set, where the average percentage increase decreased to 11.95% (28% performance increase) and in the SSD-125 instance set, with an average percentage increase of 10.91% (9% performance increase).

Considering the number of makespan evaluations as the termination criteria along with the elapsed times allows the extrapolation of results to different computing environments and/or platforms. For a better comparison of the proposed methods we can establish a fixed amount of time as a termination criteria. In this way we have an unbiased and more accurate method for determining the most effective methods. We have repeated all the evaluations by considering the following expression: termination criteria = $(n \cdot 60\,000)/(1000)$ milliseconds. This is the same as saying that we allow 60 seconds for every 1000 jobs in a given instance. For example, we will allow a maximum time of 1200 milliseconds for the 20 job instances and 30 000 milliseconds for the 500 job instances. For this comparison we shall only consider methods that rely on a stopping criteria. We should expect the results to be different from the previous evaluation, since efficient algorithms would perform better under the maximum time criterion as they can evaluate more makespans during the time allowed. The results are given in Tables 4 and 5.

As we can see, both genetic algorithms (GA and HGA) clearly outperform all other evaluated algorithms by a significant margin, independently of the instance type or size. For example, for the SSD-10 instance set, the HGA manages a 0.53% average increase over the best solution known and the GA 0.87%. The next best method is the adaptation of the genetic algorithm of Reeves (GA_REEV) which performs similarly to the adapted ILS method with average percentage increases of 1.59% and 1.60% respectively. This means that the HGA is exactly a noteworthy 300% better than the third best algorithm. Furthermore, the HGA is 64% better than the GA meaning that the hybridization is very efficient, i.e. we obtain much better results in the same allotted time. From this same set of problems we can also see that all other evaluated methods are above the 2% average percentage increase mark.

For the SSD-50, SSD-100 and SSD-125 instance sets the situation is very similar. The HGA and GA methods are the two best algorithms and the adapted GA_REEV is the third best, the differences between this GA_REEV and the HGA being 256%, 266% and 270% for the SSD-50, SSD-100 and SSD-125 instance sets, respectively. The adaptations of the tabu search and simulated annealing algorithms (SPIRIT and SA_OP) have resulted in fairly good algorithms. Furthermore, it seems that the SPIRIT algorithm has benefited greatly from the modified initialization, lagging very little behind the specific SDST flowshop RMB_GRASP method in all the instances types tested. The adapted GA_AA algorithm does not show particularly good results, despite being a very recent algorithm. Finally, Fig. 7 shows the trend of the best methods evaluated as the ratio between the setup times and the processing times increases.

7. Conclusions and future research

In this paper we have addressed the problem of scheduling a permutation flowshop problem where there are sequence dependent setup times on the machines, commonly known as the SDST flowshop. The optimisation criteria considered is the minimisation of the makespan or $C_{\max}$. We have tested an advanced genetic algorithm and an hybrid version which incorporates a special initialization of the population and a new generational scheme that enforces strong pressure but at the same time avoids premature convergence of the

Table 4
Average percentage increase over the best known upper bound for the methods evaluated with the termination criteria set at $(n \cdot 60000)/1000$ milliseconds maximum time. Instances SSD-10 and SSD-50

Instance	RANDOM	GA	ILS	SA_OP	SPIRIT	GA_REEV	HGA	RMB_GRASP	GA_AA
SSD-10 instances									
20×5	6.19	0.31	0.61	1.89	2.03	1.05	**0.15**	1.30	1.22
20×10	8.51	0.44	1.10	2.35	2.73	1.74	**0.20**	1.81	1.81
20×20	7.53	0.45	1.23	2.32	2.68	1.75	**0.30**	1.57	1.89
50×5	7.94	0.50	1.23	1.84	1.68	1.14	**0.26**	1.96	2.07
50×10	12.25	0.94	2.05	3.07	2.92	2.04	**0.50**	2.65	3.23
50×20	13.46	1.03	2.60	3.11	2.92	2.39	**0.58**	2.77	3.53
100×5	7.90	0.68	1.43	1.40	1.51	1.15	**0.28**	1.89	2.28
100×10	11.84	0.96	1.81	2.20	2.01	1.59	**0.44**	2.18	2.83
100×20	14.29	1.44	2.59	3.02	2.51	1.98	**1.11**	2.66	3.10
200×10	10.87	1.19	1.68	1.95	1.75	1.39	**0.64**	1.82	2.03
200×20	13.69	1.62	1.97	4.51	2.16	1.90	**1.13**	2.04	2.27
500×20	11.69	0.89	0.97	4.22	1.06	0.97	**0.81**	1.15	1.09
Average	10.51	0.87	1.60	2.66	2.16	1.59	**0.53**	1.98	2.28
SSD-50 instances									
20×5	11.53	1.00	2.12	4.44	4.98	3.11	**0.67**	2.82	3.67
20×10	10.07	1.06	2.03	3.56	4.67	2.86	**0.56**	2.64	2.90
20×20	7.89	0.45	1.71	2.64	3.62	2.25	**0.28**	1.97	2.37
50×5	18.18	1.86	4.84	5.25	5.73	3.77	**0.81**	5.07	5.79
50×10	16.63	2.05	4.52	5.14	5.33	3.74	**1.26**	4.71	5.47
50×20	14.38	1.77	4.33	4.10	4.58	3.44	**1.28**	3.72	4.75
100×5	20.13	2.45	5.27	4.39	4.98	3.55	**1.18**	5.16	5.66
100×10	18.06	2.53	4.75	4.37	4.07	3.47	**1.63**	4.34	5.09
100×20	15.77	2.45	4.07	4.12	3.78	3.16	**1.90**	3.66	4.21
200×10	17.55	2.59	3.59	4.34	3.31	3.00	**1.83**	3.64	3.66
200×20	15.78	2.32	3.23	6.28	3.18	3.03	**1.93**	3.05	3.25
500×20	14.16	**1.11**	1.52	6.51	1.51	1.50	1.13	1.62	1.52
Average	15.01	1.80	3.50	4.60	4.15	3.07	**1.20**	3.53	4.03

population. The used crossover operators have shown better results than the *Partially Mapped Crossover* or PMX. Other features of the tested algorithms include a restart scheme that replaces a given portion of the population if the best makespan has not been improved for a given number of generations. The genetic algorithm has been calibrated by means of a Design of Experiments (DOE) approach. We have used four instance sets of 120 instances each, based on the problems of Taillard with the addition of sequence dependent setup times.

For evaluating the performance of the two tested genetic algorithms we have proposed adaptations of some of the best performing regular flowshop algorithms to the SDST flowshop, such as the simulated annealing of Osman and Potts (1989), the tabu search of Widmer and Hertz (1989), the genetic algorithm of Reeves (1995) and the iterated local search of Stützle (1998), as well as an adaptation of a recent genetic algorithm for the no-wait flowshop (Aldowaisan and Allahverdi, 2003). We have also coded several specific methods for the SDST flowshop, such as the SETUP and TOTAL heuristics of Simons (1992) and the GRASP metaheuristic of Ríos-Mercado and Bard (1998b) among others. The obtained results show that in every instance type and in every instance size, the two proposed genetic algorithms clearly outperform all other compared algorithms. Our algorithms are between 256% and 300% better than the best existing algorithms when compared

Table 5
Average percentage increase over the best known upper bound for the methods evaluated with the termination criteria set at $(n \cdot 60000)/1000$ milliseconds maximum time. Instances SSD-100 and SSD-125

Instance	RANDOM	GA	ILS	SA_OP	SPIRIT	GA_REEV	HGA	RMB_GRASP	GA_AA
SSD-100 instances									
20×5	15.59	1.73	4.49	7.29	8.52	5.97	**0.72**	4.79	5.92
20×10	12.15	1.44	3.56	5.74	5.75	4.31	**0.67**	3.53	4.14
20×20	9.15	0.76	2.58	3.61	4.28	2.85	**0.41**	2.68	3.13
50×5	26.84	2.98	7.73	7.91	8.33	6.32	**1.79**	7.58	8.45
50×10	20.77	2.72	6.29	6.71	6.29	5.58	**1.81**	6.17	6.62
50×20	16.11	2.22	4.91	5.19	4.88	4.05	**1.58**	4.41	5.19
100×5	30.81	3.98	8.21	7.60	7.84	6.39	**2.11**	7.75	8.49
100×10	22.88	3.39	6.02	6.02	5.82	4.56	**2.18**	5.66	6.21
100×20	18.39	3.32	5.43	5.51	5.06	4.46	**2.68**	4.76	5.50
200×10	23.83	3.57	5.13	6.73	4.93	4.54	**2.64**	5.03	5.13
200×20	18.56	2.71	3.44	8.04	3.41	3.33	**2.46**	3.75	3.44
500×20	17.44	1.26	1.71	8.75	1.70	1.69	**1.25**	2.02	1.71
Average	19.38	2.51	4.96	6.59	5.57	4.50	**1.69**	4.84	5.33
SSD-125 instances									
20×5	17.17	1.61	4.47	8.06	9.30	5.73	**0.76**	5.31	6.14
20×10	12.90	1.22	3.36	6.45	5.54	4.81	**0.57**	4.02	4.55
20×20	9.28	0.87	2.52	3.62	3.79	2.89	**0.52**	2.78	2.84
50×5	30.55	3.81	9.19	10.12	9.93	7.85	**1.99**	8.85	9.72
50×10	22.52	2.94	7.29	7.51	7.64	6.29	**1.80**	6.60	7.72
50×20	17.33	2.82	5.93	6.11	5.52	4.66	**2.13**	5.22	6.18
100×5	34.62	3.96	9.25	8.48	8.67	6.65	**2.23**	8.35	9.45
100×10	25.08	3.90	6.62	6.77	5.36	5.12	**2.24**	6.04	6.79
100×20	19.41	3.34	5.36	5.68	4.81	4.28	**2.57**	4.88	5.42
200×10	26.46	3.91	5.49	7.34	5.19	4.93	**3.02**	5.58	5.49
200×20	19.59	2.79	4.07	8.49	4.00	3.86	**2.66**	3.89	4.08
500×20	18.81	**1.29**	1.82	9.52	1.81	1.80	1.32	2.21	1.82
Average	21.14	2.71	5.45	7.35	5.96	4.91	**1.82**	5.31	5.85

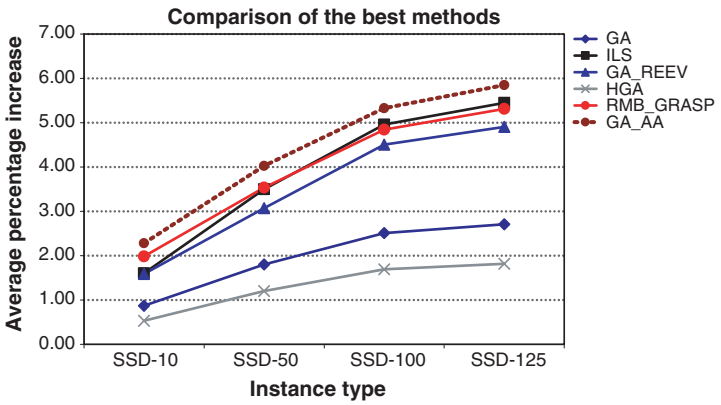

Fig. 7. Comparison of the best methods evaluated for the different instance types and the termination criteria set at $(n \cdot 60000)/1000$ milliseconds maximum time.

under the maximum elapsed time termination criteria and between 366% and 488% when compared under the maximum makespan evaluations termination criteria. The obtained results indicate that these two new genetic algorithms are the most effective methods proposed to date for the SDST flowshop problem by a significant margin.

The proposed genetic algorithms are currently being extended to take into account more realistic aspects of the problem, such as the existence of unrelated parallel machines at each stage (hybrid flowshops), due dates, machine eligibility, or machine lags.

All the code used in this paper, as well as the proposed algorithms and benchmarks are available upon request from the authors.

Acknowledgements

This work is funded by the Polytechnic University of Valencia, Spain under an interdisciplinary project and by the Spanish Department of Science and Technology (research project ref. DPI2001-2715-C02-01). The authors would also like to thank the two anonymous referees for their insight shown in their comments in a previous version of this paper.

References

Alcaraz, J., Maroto, C., 2001. A robust genetic algorithm for resource allocation in project scheduling. Annals of Operations Research 102, 83–109.

Alcaraz, J., Maroto, C., Ruiz, R., 2003. Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. Journal of the Operational Research Society 54 (6), 614–626.

Aldowaisan, T., Allahverdi, A., 2003. New heuristics for no-wait flowshops to minimize makespan. Computers and Operations Research 30, 1219–1231.

Allahverdi, A., Gupta, J.N.D., Aldowaisan, T., 1999. A review of scheduling research involving setup considerations. OMEGA, The international Journal of Management Science 27, 219–239.

Campbell, H.G., Dudek, R.A., Smith, M.L., 1970. A heuristic algorithm for the n job, m machine sequencing problem. Management Science 16 (10), B630–B637.

Chen, C.-L., Vempati, V.S., Aljaber, N., 1995. An application of genetic algorithms for flow shop problems. European Journal of Operational Research 80, 389–396.

Cheng, T.C.E., Gupta, J.N.D., Wang, G., 2000. A review of flowshop scheduling research with setup times. Production and Operations Management 9 (3), 262–282.

Conway, R.W., Maxwell, W.L., Miller, L.W., 1967. Theory of Scheduling. Addison-Wesley, Reading, MA.

Corwin, B.D., Esogbue, A.O., 1974. Two machine flow shop scheduling problems with sequence dependent setup times: A dynamic programming approach. Naval Research Logistics Quarterly 21, 515–524.

Das, S.R., Gupta, J.N.D., 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, 1365–1373.

Garey, M.R., Johnson, D.S., Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1 (2), 117–129.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics 5, 287–326.

Gupta, J.N.D., 1975. A search algorithm for the generalized flowshop scheduling problem. Computers & Operations Research 2, 83–90.

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

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

Johnson, S.M., 1954. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1, 61.

Michalewicz, Z., 1996. Genetic Algorithms + Data Structures = Evolution Programs, third ed. Springer-Verlag, Berlin.

Murata, T., Ishibuchi, H., Tanaka, H., 1996. Genetic algorithms for flowshop scheduling problems. Computers and Industrial Engineering 30 (4), 1061–1071.

Nawaz, M., Enscore Jr., E.E., Ham, I., 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA, The International Journal of Management Science 11 (1), 91–95.

Osman, I.H., Potts, C.N., 1989. Simulated annealing for permutation flow-shop scheduling. OMEGA, The International Journal of Management Science 17 (6), 551–557.

Parthasarathy, S., Rajendran, C., 1997. 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, 255–263.

Pinedo, M., 2002. Scheduling: Theory, Algorithms and Systems, second ed. Prentice-Hall, New Jersey.

Reeves, C.R., 1995. A genetic algorithm for flowshop sequencing. Computers and Operations Research 22 (1), 5–13.

Ríos-Mercado, R.Z., Bard, J., 1998a. Computational experience with a branch-and-cut algorithm for flowshop scheduling

with setups. Computers and Operations Research 25 (5), 351–366.

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

Ríos-Mercado, R.Z., Bard, J., 1999a. A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. IEE Transactions 31, 721–731.

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

Ruiz, R., Maroto, C., 2004. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research (to appear).

Ruiz, R., Maroto, C., Alcaraz, J., 2003. Two new robust genetic algorithms for the flowshop scheduling problem. Technical Report, Universidad Politécnica de Valencia, Valencia, España, Grupo de Investigatión Operativa GIO. OMEGA, The International Journal of Management Science (submitted).

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

Srikar, B.N., 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., 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, TU Darmstadt, AIDA-98-04, FG Intellektik.

Szwarc, W., Gupta, J.N.D., 1987. A flow-shop problem with sequence-dependent additive setup times. Naval Research Logistics Quarterly 34, 619–627.

Taillard, E., 1990. Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research 47, 67–74.

Taillard, E., 1993. Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285.

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

Vallada, E., Ruiz, R., Maroto, C., 2003. Synthetic and real benchmarks for complex flow-shop problems. Technical Report, Universidad Politécnica de Valencia, Valencia, España, Grupo de Investigatión Operativa GIO.

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

Yang, W.-H., Liao, C.-J., 1999. Survey of scheduling research involving setup times. International Journal of Systems Science 30 (2), 143–155.