Available online at www.sciencedirect.com

SCIENCE DIRECT®

European Journal of Operational Research 169 (2006) 781–800

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility

Rubén Ruiz [*], Concepción Maroto

Departamento de Estadística e Investigación Operativa Aplicadas y Calidad, Universidad Politécnica de Valencia, Camino de Vera S/N, 46021 Valencia, Spain

Received 30 September 2003; accepted 28 June 2004
Available online 14 March 2005

Abstract

After 50 years of research in the field of flowshop scheduling problems the scientific community still observes a noticeable gap between the theory and the practice of scheduling. In this paper we aim to provide a metaheuristic, in the form of a genetic algorithm, to a complex generalized flowshop scheduling problem that results from the addition of unrelated parallel machines at each stage, sequence dependent setup times and machine eligibility. Such a problem is common in the production of textiles and ceramic tiles. The proposed algorithm incorporates new characteristics and four new crossover operators. We show an extensive calibration of the different parameters and operators by means of experimental designs. To evaluate the proposed algorithm we present several adaptations of other well-known and recent metaheuristics to the problem and conduct several experiments with a set of 1320 random instances as well as with real data taken from companies of the ceramic tile manufacturing sector. The results indicate that the proposed algorithm is more effective than all other adaptations.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Scheduling; Hybrid flowshop; Setup times; Genetic algorithm

1. Introduction

The area of flowshop scheduling has been a very active field of research during the last 50 years since Johnson (1954) seminal work. This research includes literally hundreds of papers in exact techniques and also heuristic and metaheuristic algorithms for flowshop scheduling problems and some of its variants. However, the reality of the production systems is more complicated and there is a noticeable gap between the theory and the application of the existing methods. Graves (1981) pointed out this problem and proposed several research directions to aid in bridging this gap. Other studies, like the ones in Ledbetter and Cox (1977) and Ford et al. (1987), show that there is

[*] 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 © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2004.06.038

very little application in practice of the existing scheduling methods. The work of McKay et al. (1988) also cites several practical situations which existing scheduling research does not solve. Later, Olhager and Rapp (1995) stated the same: Proposed methods are not easy to use and they do not take into account several important aspects present in most production systems. Additionally, there are many papers in which the authors explicitly state the necessity of proposing methods for real scheduling problems, for example in Dudek et al. (1992), MacCarthy and Liu (1993) or more recently McKay et al. (2002). Reviews of the state of the art for different scheduling environments and more specifically in the area of flowshop scheduling also have similar conclusions (see Allahverdi et al., 1999; Linn and Zhang, 1999 or Vignier et al., 1999).

In this paper we propose the adaptation of a genetic algorithm metaheuristic that performed well in regular flowshops to a much more realistic version of the problem where we find sequence dependent setup times as well as unrelated parallel machines at each production stage and machine eligibility. This problem is found in a very important industrial sector of the Spanish economy, which is the ceramic tile manufacturing sector. The proposed method has been implemented in production scheduling software, which can also be used to solve several other scheduling problems such as the production of fabric in textile companies or synthetic paints.

The rest of the paper is organized as follows: In Section 2 we provide a description of the ceramic tile manufacturing process, along with a characterization of the related production programming problem. Section 3 gives an up to date comprehensive review of the existing literature. In Section 4 we present the new genetic algorithm in detail, along with a complete calibration by means of the Design of Experiments approach. Section 5 provides several adaptations of other metaheuristics to the considered problem, as well as two complete evaluations, one with a set of random problems and the other with real problems taken from companies in the ceramic tile sector. Finally, in Section 6 some conclusions on this study are given.

2. The production process of ceramic tiles

Basically, a ceramic tile is a slab of kiln fired clay. This modest definition conceals a sophisticated manufacturing process that nowadays allows ceramic tile factories to mass produce tiles with great efficiency. The most widely used manufacturing process is called "fast single-firing" and can be briefly explained as follows: The clay powder is loaded into molds and it is hydraulically pressed to form what is called the bisque. The bisque is then run through a horizontal dryer to reduce its water content and then various types of glazes are applied in the glazing lines. The different glaze applications determine the style, color, surface and characteristics of the tile. After this process, the glazed bisque is transported to an oven, called a kiln, where the controlled firing curve causes chemical reactions both in the bisque and in the glaze. After being fired, the tile is loaded onto the selection line where the tiles are sorted by tones and classes. Basically, from start to finish, the product is never touched, as the transport between the different machines in the shop is done by Automated Guided Vehicles (AGVs).

To further understand this production problem we have conducted a comprehensive survey in 87 companies of the Spanish ceramic tile sector, which represents 38% of the existing firms. The results of the study can be seen in Vallada et al. (2005) and the salient conclusions are now summarized:

- All the different tile types follow the same flow in the shop.
- There are several multi-purpose machines available at each production stage.
- All the products need setups in the molds, glazing lines, kilns and classification lines prior to production. These setup times are sequence dependent.
- Not all products can be processed by all machines at a given production stage. Furthermore, the parallel machines at each stage are unrelated.
- Very few companies have software to schedule production and those who have do agree that the available software has many shortcomings.

- The total production time is a very important factor for many of the firms consulted.

With the above results we can characterize the production problem as a flowshop with unrelated parallel machines at each stage, commonly known as a hybrid flowshop. Since the total production time is important, we choose to use makespan ($C_{\max}$) minimization as the scheduling criterion. The number of stages varies from two in the most rigid production systems to five or more in the biggest companies. The number of unrelated parallel machines at each stage ranges between one and three, although there are examples of big production centers with more machines. Normally the production managers schedule between 30 and 50 production orders per week with exceptions arising in big factories where more than 100 orders are scheduled. We can also see that there are sequence dependent setup times per machine and stage. Since not all products can be processed on all available machines, we have what is called machine eligibility.

In a hybrid flowshop we have a set N of jobs, $N = \{1,\ldots,n\}$, that have to be processed in a set of M stages, $M = \{1,\ldots,m\}$. At every stage i, $i \in M$, we have a set $M_i = \{1,\ldots,m_i\}$ of unrelated parallel machines that can process the jobs where $|M_i| \geqslant 1$. Every job has to pass through all stages and must be processed by exactly one machine at every stage. $p_{i_l,j}$ indicates the processing time of job j, $j \in N$, at machine l, $l \in M_i$, inside stage i. We also have a machine based sequence dependent setup time in every machine l at stage i when processing job k, $k \in N$, after having processed job j which is noted as $S_{i_l,j,k}$. Finally, for every job j and stage i we have a set E_{ij} of eligible machines that can process job j. Clearly $1 \leqslant |E_{ij}| \leqslant m_i$ and we have observed a tendency in the studied real problems where many times for a given job j either $|E_{ij}| = 1$ or $|E_{ij}| = m_i - 1$, i.e. some jobs can be processed only on one machine at a given stage i or on all machines but one. It is important not to confuse the hybrid flowshop with the flowshop with multiple processors (FSMP) or the flexible flow line (FFL) problem. In these two latter problems the machines available at each stage are identical.

The hybrid flowshop problem is significantly more complex than the regular flowshop. Gourgand et al. (1999) showed the total number of possible solutions to be $n!(\prod_{i=1}^{m} m_i)^n$. Moreover, Gupta (1988) showed the FSMP problem with only two stages ($m = 2$) to be $\mathcal{NP}$-Hard even when one of the two stages contains a single machine. Since the FSMP problem can be seen as a specific case of the hybrid flowshop, we can conclude then that this latter problem is also $\mathcal{NP}$-Hard.

Considering the well-known three field notation $\alpha/\beta/\gamma$ for scheduling problems and the extension for hybrid flowshops proposed by Vignier et al. (1999), the real production problem considered here can be noted as $FHm, ((RM^{(i)})_{i=1}^{(m)})/S_{sd}, M_j/C_{\max}$. It is important to remark that this problem is not a special case of the regular flowshop but a much more complex generalization. For example, if we only consider one machine per stage (therefore dropping also the machine eligibility constraints) then we have a flowshop with sequence dependent setup times or SDST flowshop ($F/S_{sd}/C_{\max}$).

3. Literature review

One of the earliest works that deal with problems related to the one considered in this paper is by Arthanary and Ramaswamy (1971), where the authors proposed a branch and bound algorithm (B&B) for a simple FSMP problem with only two stages. Salvador (1973) showed dynamic programming algorithms for the no-wait FSMP. The algorithm presented can only be applied to very small instances. Other exact approaches were proposed by many authors; Brah and Hunsucker (1991) proposed a B&B algorithm for the FSMP problem with m stages and any number of identical machines per stage. However, the algorithm is only suitable for small instances and even in this case, the CPU time needed is very long. Rajendran and Chaudhuri (1992a) developed a B&B algorithm for the FSMP but with the $F_{\max}$ optimization criterion. In a similar paper (see Rajendran and Chaudhuri, 1992b), the same authors studied the same problem but with the more common $C_{\max}$ criterion. The interesting aspect of this work is that the authors apply a B&B algorithm to the regular flowshop and with the production sequence obtained as a result they assign at each stage the jobs to the first available machine. Gupta et al. (1997) considered the two stage FSMP with only one machine in the second stage. The authors proposed a B&B algorithm as well as four heuristic methods. In Portmann et al. (1998) a B&B algorithm similar to that of Brah and Hunsucker is proposed. The main difference is that a Genetic Algorithm (GA) is used in the branching phase of the B&B method. The authors showed the resulting algorithm to be superior to that of Brah and Hunsucker. Another B&B algorithm is presented in Riane et al. (1998). In this case a real industrial problem is solved. This problem is a FSMP with three stages and the authors proposed also a dynamic programming-based heuristic.

Heuristic methods have also been profusely developed. Gupta (1988) proposed heuristic techniques for a simplified FSMP with two stages and only one machine in the second stage. The proposed heuristics are based on extensions of Johnson's well-known rule. So too are Sriskandarajah and Sethi's (1989) proposed list and Johnson's rule based algorithms for problems which also have two stages. More recently, Soewnadi and Elmaghraby (2001) have proposed several heuristics for the three stage FSMP and the $C_{\max}$ criterion. Some authors have studied the application of priority rules to several FSMP-related problems. For example Gupta and Tunc (1991) showed two priority rule-based heuristics for solving the symmetric problem of that stated in Gupta (1988). Guinet and Solomon (1996) published an interesting work about the FSMP problem. The authors also separate the sequencing from the assignment and consider the criteria of $C_{\max}$ and $T_{\max}$. For sequencing the authors use effective regular flowshop heuristics like NEH (Nawaz et al., 1983) or CDS (Campbell et al., 1970) and then the assignment of jobs to machines at each stage is done by using priority rules. A similar study is due to Santos et al. (1996) where the same solution is used.

Metaheuristic algorithms have also been proposed, for example, Voss (1993) presented a heuristic method improved with Tabu Search (TS) for the two stage FSMP where the second stage contains only one machine and where stage-based sequence dependent times are considered. In Nowicki and Smutnicki (1998) the authors propose also a TS method for the FSMP problem. The algorithm uses the critical block concept to obtain a fast and effective metaheuristic. Gourgand et al. (1999) presented several Simulated Annealing (SA)-based algorithms for the hybrid flowshop problem. A specific neighborhood is used and the authors apply the methods to a realistic industrial problem. Andrés (2001), proposed some heuristic and metaheuristic methods for the three stage FSMP problem with stage-based sequence dependent setup times.

The flexible flow line problem (FFL) is essentially similar to the FSMP, the main difference being that in the FFL the jobs might skip some stages. Wittrock (1985) proposed a heuristic for the FFL considering the minimization of the work-in-progress (WIP) criterion. Leon and Ramamoorthy (1997) developed an adaptable problem-space-based search method for the FFL with the criteria of $C_{\max}$ and $\overline{T}$. Kurz and Askin (2003) proposed three heuristics for the FFL problem and the $C_{\max}$ criterion. The authors considered stage-based sequence dependent setup times. In a recent paper (Kurz and Askin, in press) the same authors have showed several heuristics for the same problem, among them are greedy algorithms, insertion heuristics based on the Travelling Salesman Problem (TSP) and the Random Keys Genetic Algorithm.

In the literature there are some studies where real problems are considered, for example Sherali et al. (1990) worked on a two stage problem in the paper industry. As far as we know, this is the first paper that considers sequence dependent setup times in an FSMP environment. The authors separated the problem into sequencing first, and assignment of jobs to machines at each stage second, and proposed several mixed integer programming models to solve the problem. The objective function considered is a complex minimization of production costs. Another real case is shown in Guinet (1991), where several algorithms are proposed to solve the fabric production problem. The author also divides the problem in several subproblems that are independently solved by heuristics constructed for each specific case. The considered objective functions are the $C_{\max}$ and also the minimization of the average delay ($\overline{T}$). Adler et al. (1993) developed a complex system called BPSS for production programming in a company that manufactures paper products. The production problem contains setup times and in some stages unrelated parallel machines. The system is specific for the considered problem and is based on the application of priority rules. Another real application can be found in the work of Aghezzaf et al. (1995). In this case the authors propose several methods to solve a problem in the carpet manufacturing industry where they consider three stages and sequence dependent setup times. The solution is based on problem decomposition, heuristics and MIP models. Table 1 shows a summary of the different papers reviewed with their main characteristics.

From this review and from the one by Vignier et al. (1999), we can see that many papers deal with simplified cases of the FSMP with only two or three stages. We only know about three studies (Adler et al., 1993; Aghezzaf et al., 1995; Gourgand et al., 1999) where unrelated parallel machines are considered, and only two of these cases take into account sequence dependent setup times. Additionally, in Adler et al. (1993) a very specific problem is solved and the methods proposed can be considered ad hoc. Also, in Aghezzaf et al. (1995) the authors do not provide details on the methods proposed and they can also be considered as problem-specific. To the best of our knowledge, no general method has been proposed for the hybrid flowshop with unrelated parallel machines and sequence dependent setup times, let alone machine eligibility. For this reason we have developed a GA for this complex problem.

4. Genetic algorithm for the hybrid flowshop

In Ruiz et al. (2005, in press) the authors proposed several genetic algorithms for solving the regular and the SDST flowshops respectively. The good results obtained allow us to think of an adaptation of some of these algorithms for the real problem considered in this paper.

A genetic algorithm differs from other metaheuristics in the fact that it works with a set of encoded solutions to the problem, called *population*. Every solution or *chromosome* in the population is evaluated and assigned a *fitness* value. The idea is that the best individuals should also be the fittest. In a GA the population *evolves* over *generations* until some stopping criterion is reached. A generation begins with the *selection* mechanism that randomly picks some members from the population. The fittest chromosomes should have a greater chance of being selected. Then, these individuals mate in a process called *crossover* and generate some new individuals, which are often called *offspring*. Afterwards, some offspring might undergo another process called *mutation* in which some of the parts of the chromosomes or *genes* can change as in natural species mutation. Lastly, the new population is evaluated again and the whole process is repeated (see Michalewicz, 1996).

The effectiveness of a GA depends on the choice of its operators and parameters; that is, encoding, selection, population size, crossover, mutation and their probabilities. In the following, we explain how these operators and parameters are set.

4.1. Encoding, evaluation and initialization of the population

For the regular flowshop a simple permutation of the jobs in an array constitutes the most widely used encoding for the chromosomes. This is also true for the SDST flowshop (see Ruiz et al., 2005). Normally, for the hybrid flowshop we would need a different encoding to reflect the assignment of jobs to machines at every stage. However, initial versions of the GA with such a complex encoding yielded poor results. From Section 3 we have seen that many authors separate sequencing and assignment decisions in the hybrid flowshops or FSMP problems (Sherali et al., 1990; Rajendran and Chaudhuri, 1992a; among others). This is usually done by obtaining a sequence with an effective permutation flowshop heuristic first and then assigning jobs to machines at every stage by priority rules or to the first available machine after having obtained the sequence. In this paper we follow a different path. We choose to maintain a simple

Table 1
Reviewed papers for FSMP and hybrid flowshop

Year	Author/s	Problem	Comments
1971	Arthanary and Ramaswamy	$FH2, ((PM^{(1)}), (P1^{(2)}))//C_{\max}$	B&B, only for small instances
1973	Salvador	$FHm, ((PM^{(i)})_{i=1}^{(m)})/nwt/C_{\max}$	DP, only for small instances
1988	Gupta	$FH2, ((P2^{(1)}), (P1^{(2)}))//C_{\max}$	Heuristics based on Johnson's rule
1989	Sriskandarajah and Sethi	$FH2, (((PM^{(i)})_{i=1}^{(2)})//C_{\max}$	List-based heuristics
1990	Sherali et al.	$FH2, (((PM^{(i)})_{i=1}^{(2)})/S_{sd}/Costs$	Specific problem, MIP models
1991	Guinet	$FHm, (((PM^{(i)})_{i=1}^{(m)})/S_{sd}/C_{\max}, \overline{T}$	Specific problem, ad hoc heuristics
	Gupta and Tunc	$FH2, ((P1^{(1)}), (P2^{(2)}))//C_{\max}$	Priority rule-based heuristics
	Brah and Hunsucker	$FHm, ((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	B&B, only for small instances
1992	Rajendran and Chaudhuri	$FHm, ((PM^{(i)})_{i=1}^{(m)})//F_{\max}$	B&B, only for small instances
	Rajendran and Chaudhuri	$FHm, ((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	B&B, regular flowshop based
1993	Adler et al.	$FHm, ((RM^{(i)})_{i=1}^{(m)})/S_{sd}/\sum_{j=1}^{n} w_j T_j$	Specific problem, priority rules
	Voss	$FH2, ((PM^{(1)}), (P1^{(2)}))/S_{sd}/C_{\max}$	Heuristics and tabu search
1995	Aghezzaf et al.	$FHm, ((RM^{(i)})_{i=1}^{(m)})/S_{sd}/C_{\max}$, $F_{\max}$ and $\overline{F}$	Specific problem, ad hoc heuristics
1996	Guinet and Solomon	$FHm, ((PM^{(i)})_{i=1}^{(m)})//C_{\max}, T_{\max}$	Heuristics and priority rules
	Santos et al.	$FHm, ((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	Heuristics and priority rules
1997	Gupta et al.	$FH2, ((PM^{(1)}), (P1^{(2)}))//C_{\max}$	Heuristics and B&B
	Leon and Ramamoorthy	$FFm, ((PM^{(i)})_{i=1}^{(m)})//C_{\max}, \overline{T}$	Problem-space search, FFL
1998	Nowicki and Smutnicki	$FHm, ((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	TS based on critical path
	Portmann et al.	$FHm, ((PM^{(i)})_{i=1}^{(m)})//C_{\max}$	B&B mixed with GAs
	Riane et al.	$FH3, ((P1^{(1)}), (P2^{(2)}), (P1^{(3)}))//C_{\max}$	B&B, DP
1999	Gourgand et al.	$FHm, ((RM^{(i)})_{i=1}^{(m)})//C_{\max}$	SA-based heuristics
2001	Soewnadi and Elmaghraby	$FH3, ((PM^{(i)})_{i=1}^{(3)})//C_{\max}$	Heuristics
	Andrés	$FH3, ((P4^{(1)}), (P2^{(2)}), (P3^{(3)}))/S_{sd}/C_{\max}$	Heuristics and metaheuristics
2003	Kurz and Askin	$FFm, ((PM^{(i)})_{i=1}^{(m)})/S_{sd}/C_{\max}$	Heuristics, FFL
	Kurz and Askin	$FFm, ((PM^{(i)})_{i=1}^{(m)})/S_{sd}/C_{\max}$	Heuristics and metaheuristics, FFL

permutation of the jobs indicating the order in which the jobs are processed in the shop at every stage. However, in our case the assignment of jobs to machines at every stage is done in the evaluation function of the chromosomes, that is, the fitness value of the individuals already contains a $C_{\max}$ that is calculated taking into account the assignment and not only the sequence of jobs.

This assignment is done differently to that which can be found in previous research where jobs are assigned to the first available machine (Rajendran and Chaudhuri, 1992a; Guinet and Solomon, 1996). In an FSMP with no setup times and no eligibility constraints the first available machine would also result in the earliest completion time of the job. In a hybrid flowshop with unrelated parallel machines we can find that the first available machine is very slow for a given job and thus assigning the job to this machine can result in a later completion time compared to other machines. With the addition of setup times this problem can get even worse. To solve the problem we change the way the $C_{\max}$ is calculated and jobs are assigned to the machine that can finish the job at the earliest time in a given stage, taking into consideration different processing speeds, setup times and machine eligibility. The calculation of this $C_{\max}$ is as follows:

Let us have a permutation or sequence π of n jobs. The job in the jth position in the sequence is denoted as $\pi_{(j)}$, $j \in N$. Every job has to be processed at every stage, so we will have m tasks per job. We denote with $C_{i,\pi_{(j)}}$ the completion time of job $\pi_{(j)}$ at stage i, where $i \in M$. We refer to L_{i_l} as the last job that was assigned to machine l within stage i, $l \in M_i$. We proceed by considering all jobs in π in order and for each job we go through all m stages looking for the machine that can finish the job at the earliest time taking into consideration the different processing times, sequence dependent setup times and machine eligibility. So for every job $\pi_{(j)}$ and for every stage i we calculate the following expression:

$$C_{i,\pi_{(j)}} = \min_{\substack{l=1 \\ l \in E_{ij}}}^{m_i} \{\max\{C_{i,L_{i_l}} + S_{i_l,L_{i_l},\pi_{(j)}}; C_{i-1,\pi_{(j)}}\} + p_{i_l,\pi_{(j)}}\}, \tag{1}$$

where initially $L_{i_l} = C_{0,\pi_{(j)}} = C_{i,0} = S_{i_l,0,\pi_{(j)}} = 0$. Notice that in a given stage i we only consider the machines $l \in M_i$ that can process job $\pi_{(j)}$, i.e. the machines $l \in E_{ij}$. Also, $S_{i_l,L_{i_l},\pi_{(j)}}$ represents the setup time of machine l at stage i when processing job $\pi_{(j)}$ after having processed the previous work assigned to this machine l (L_{i_l}). Once all jobs are assigned to machines in all stages we calculate the makespan as follows:

$$C_{\max} = \max_{j=1}^{n}\{C_{m,\pi_{(j)}}\}. \tag{2}$$

With these calculations we have a genetic algorithm that works with the permutation of jobs as an encoding of the solutions but that considers the assignment of jobs to machines at every stage in the $C_{\max}$ and therefore in the fitness value, in this way the algorithm considers both the sequencing and the assignment simultaneously.

The population is formed by a given number of individuals or chromosomes denoted by P_{size}. For the initialization we generate all these P_{size} individuals at random except one that is obtained by making use of the well-known NEH heuristic for the permutation flowshop problem. Moreover, to take into consideration all the constraints in the considered problem we change the $C_{\max}$ evaluation in every step of the NEH heuristic with the calculations stated above, this way we have an NEH-modified heuristic that gives a feasible and good starting individual for the population. This new version of the NEH heuristic will be referred to as NEH$_H$.

4.2. Selection, crossover and mutation operators

For the selection we simply use two of the most well-known selection schemes, which are ranking and stochastic binary tournament selection (see Michalewicz, 1996).

The crossover operator generates offspring by coalescing two other sequences which we will call parents. The objective is to generate new solutions with better $C_{\max}$ values after crossover. There exist many crossover operators that can be used for the permutation encoding. In Ruiz et al. (in press) the authors tested many crossover operators for the regular flowshop (PMX or *Partially Mapped Crossover*, OP or *One Point Order Crossover*, TP or *Two Point Order Crossover*, OX or *Order Crossover*, UOB or *Uniform Order Based* and several others). The results showed that the offspring generated after crossover tended to be worse than their progenitors on many occasions. This behaviour was due to the fact that crossover tends to break building blocks, specially in the latter stages of the algorithm. We also found the same results for the SDST flowshop (see Ruiz et al., 2005). In this work we have proposed four new crossover operators for the regular flowshop that resulted in better effectiveness of the genetic algorithm. These operators identify and maintain building blocks in the crossover, considering building blocks as similar job occurrences in both parents. Building blocks are copied over to offspring unaltered.

The four proposed crossover operators are *Similar Job Order Crossover* or SJOX, *Similar Block Order Crossover* or SBOX, *Similar Job 2-Point Order Crossover* or SJ2OX and *Similar Block 2-Point Order Crossover* or SB2OX. The SJOX operator proceeds as follows: First both parents are examined on a position-by-position basis and the building blocks of jobs are directly copied to the offspring (Fig. 1a). Then, a random cut point is drawn and the section before the point is directly

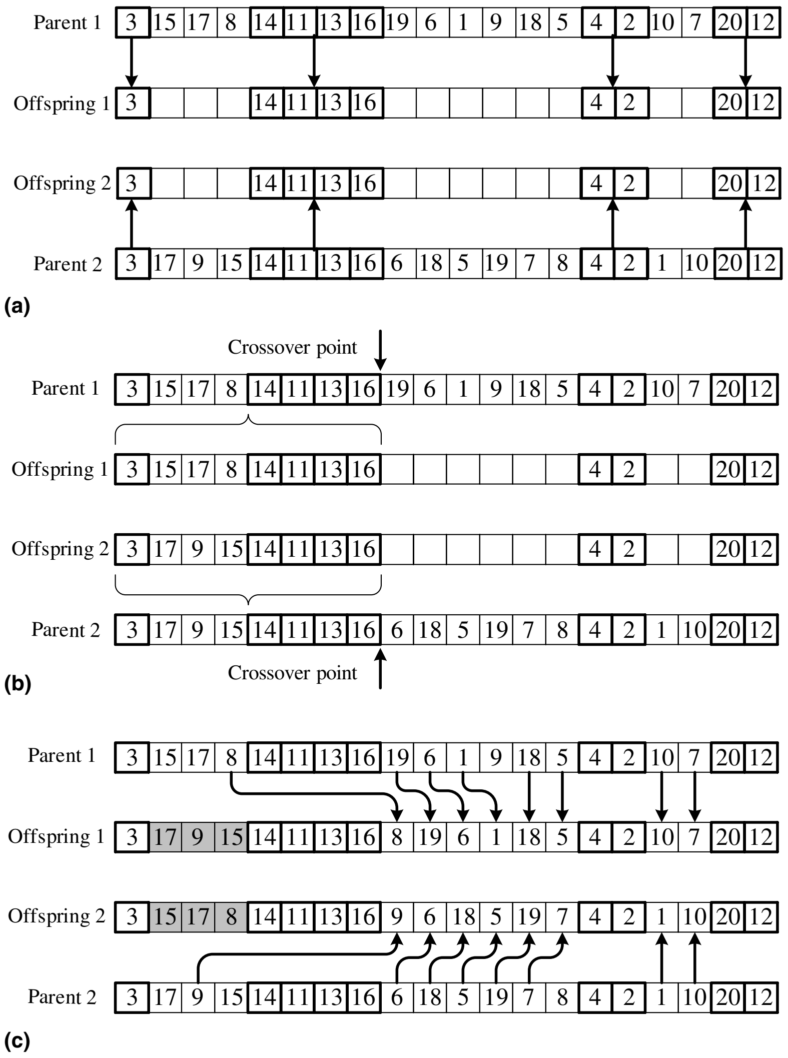

Fig. 1. "Similar Job Order Crossover" (SJOX): (a) first, the common jobs in both parents are copied over to the offspring; (b) then, jobs before a randomly chosen cut point are inherited from the direct parent; (c) the missing elements in the offspring are copied in the relative order of the other parent.

copied to the offspring (Fig. 1b). Lastly, the missing elements of each offspring are copied in the relative order of the other parent (Fig. 1c) so as to maintain feasibility in the job permutation.

In the SBOX the first step of the SJOX is modified by considering blocks of at least two consecutive identical jobs and only those two-job blocks on both parents are directly copied to the offspring. The SJ2OX and SB2OX are very similar to SJOX and SBOX respectively, the only difference is that in the second step, two random cut points are drawn and the section between these two points is directly copied to the offspring.

Most genetic algorithms incorporate a mutation operator mainly to avoid convergence to local optima in the population and for recovering lost genetic material. In the proposed genetic algorithm we use the SHIFT mutation, where a randomly picked position in the sequence is relocated at another random position and the jobs between these two positions move along. For example, if we remove job at position 3 of the sequence and we want to reinsert it at position 7, jobs at positions from 4 to 7 slide and occupy positions from 3 to 6. This procedure has been regarded in various studies as the best mutation operator for flowshop problems (see Reeves, 1995; Murata et al., 1996).

4.3. Restart and generational schemes

To further avoid premature convergence in the population we have implemented two methods, named restart and generational schemes. The restart scheme is based on a previous work (see Alcaraz et al., 2003) and is similar to the one implemented in Ruiz et al. (in press). This method works as follows: At each generation we store the minimum makespan. If in the following generation the minimum makespan has not changed, we increment a counter. When this counter, called *countmak*, becomes higher than a control parameter called G_r, the following procedure is applied:

- Create a sorted list $\wp$ of P_{size} elements with the $C_{\max}$ of the chromosomes in ascending order.
- Skip the chromosomes in the first 20% of elements of $\wp$, i.e. elements $1, 2, \ldots, \lceil 0.2 \cdot Psize \rceil$.
- The remaining 80% of chromosomes in the sorted list ($\lceil 0.2 \cdot Psize \rceil + 1, \lceil 0.2 \cdot Psize \rceil + 2, \ldots, P_{size}$) are disregarded and re-generated in the following way:
  – Half of these new chromosomes ($\lceil 0.2 \cdot Psize \rceil + 1, \lceil 0.2 \cdot Psize \rceil + 2, \ldots, \lceil 0.6 \cdot P_{size} \rceil$) are generated by copying a randomly chosen chromosome from the first 20% of $\wp$. This new copied chromosome is mutated once with the SHIFT mutation.
  – The other half ($\lceil 0.6 \cdot Psize \rceil + 1, \lceil 0.6 \cdot Psize \rceil + 2, \ldots, P_{size}$) are completely new random chromosomes.
- Make $countmak = 0$.

With this procedure, whenever the lowest makespan in the population does not change for more than G_r generations, the restart procedure replaces the 80% worst individuals of the population with both new good chromosomes and new random genetic material.

The procedure by which new individuals replace old members from the previous generation is called generational scheme. As was shown in Reeves (1995), Ruiz et al. (2005, in press), a steady state GA where the offspring replace the worst individuals in the population yielded much better results than regular GAs. To avoid premature convergence, the proposed generational scheme does not allow for clones in the population, i.e. a new individual will only replace the worst individual in the population if its makespan is better than that of the worst and if its sequence is not already in the population.

4.4. Experimental calibration of the genetic algorithm

In this section we aim at carrying out extensive experiments to correctly calibrate the algorithm which we will call GA_H. We have done several full factorial experimental designs where all possible combinations of the following factors are tested:

- Selection type: 2 levels (Tournament and Ranking)
- Crossover type: 9 levels (PMX, OP, OX, UOB, TP, SJOX, SBOX, SJ2OX and SB2OX)
- Crossover probability (P_c): 6 levels (0.0, 0.1, 0.2, 0.3, 0.4 and 0.5)
- Mutation probability (P_m): 5 levels (0.0, 0.005, 0.01, 0.015 and 0.02)
- Population size (P_{size}): 4 levels (20, 30, 40 and 50)
- Restart (G_r): 3 levels (25, 50 and 75)

All cited factors result in a total of $2 \cdot 9 \cdot 6 \cdot 5 \cdot 4 \cdot 3 = 6480$ different combinations and algorithms. For each one of them we aim at solving a full set of 1320 problems that can be described as follows: There are four different configurations for the number of jobs (n), this is, $n = (20, 50, 100, 200)$, and three for the number of stages (m), $m = (5, 10, 20)$. All the combinations of n and m are considered except 200×5, therefore we have 11 different configurations. For each one of them we have 10 different problems where the processing times are uniformly distributed in the range [1, 99]. Over these problems we also define three main groups of instances where we set a given number of machines per stage. In the first one we have a random uniformly distributed number of machines of between one and three machines per stage. In the second group we have a fixed number of two machines per stage and in the third a total of three machines per stage. Inside each group we also define four different subgroups of problems with different configurations as regards the sequence dependent setup times. The setup times are drawn from the uniform distributions $U[1, 9]$, $U[1, 49]$, $U[1, 99]$ and $U[1, 124]$ respectively for each subgroup. Thus, on average, they correspond to 10%, 50%, 100% and 125% of the average processing times. As for the machine eligibility, for all 12 sets and for all jobs in all instances there is a 25% probability of setting a given $p_{i_{lj}} = -1$, which indicates that machine l in stage i is not eligible for processing job j. Of course the instances are generated so that for every job j and stage i, $|E_{ij}| \geqslant 1$. Summing up, we have three different configurations for the number of machines per stage and inside each one four configurations for the setup times. This gives an overall of 12 different sets with 110 problems each of which cover many different possible combinations. The first set is referred to as SSD10_P13 where we have 110 problems as stated above with between 1 and 3 machines per state and where the setup times are, on average, 10% the processing times. The biggest 200 jobs and 20 stage problems with three machines per stage give a total of 60 machines. Notice that the setup matrices are machine based, not stage based. That means that for this biggest problems we have 60 matrices of size 200×200 with setup times and one matrix of 200×60 with processing times which constitutes an enormous amount of data.

We have conducted 12 different experimental designs, with the same mentioned experimental setup, one for each instance group (110 problems) and have evaluated all 6480 algorithms alternatives with the stopping criterion set at 8000 makespan evaluations. The response variable is based on the following performance measure:

$$\%\text{Increase Over the Best Solution} = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \cdot 100,$$

where Heu_{sol} is the best makespan obtained by a given algorithm and $Best_{sol}$ is the makespan obtained for each instance after 500,000 makespan evaluations of the proposed GA with standard parameters (Ranking selection, OP crossover, $P_c = 0.6$, $P_m = 0.01$, $P_{size} = 50$ and $G_r = 50$). The response variable is then the average of all 110 problems. All experiments were performed in a cluster of 4 PC/AT computers with Athlon XP 1600+ processors and 512 MBytes of main memory.

The resulting experiments were analyzed by means of a multifactor Analysis of Variance (ANOVA) technique. As regards the ANOVA's models adequacy to the data we can say that all three hypotheses (normality, homogeneity of variance or homocedasticity and independence of the residuals) were accepted in all 12 experiments (see Montgomery, 2000). All the experiments were carried out at 95% confidence level. We are going to comment on the results for the first experiment, called SSD10_P13 where we have 110 instances with between one and three machines per stage and setup times that are 10% the processing times.

The most influential factor turned out to be the type of selection. To choose the best levels for the studied factors we can use means plots to graphically see which level is best for the genetic algorithm. The two means for the type of selection are plotted in Fig. 2.

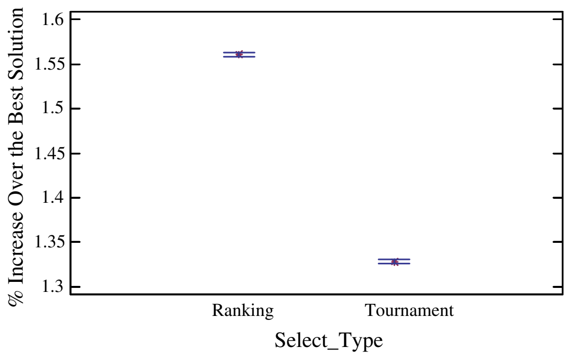

Fig. 2. Means and 95.0% LSD intervals plot for the type of selection factor (`Select_Type`), experiment SSD10_P13.

We can see that there is a clear statistically significant difference between the two types of selection, tournament selection being clearly better (the non-overlapping intervals indicate a statistically significant difference at the 95% confidence level). This result can be explained by the fact that tournament selection imposes a greater selection pressure in the algorithm than ranking since the best individuals are chosen more frequently. This could have resulted in premature convergence if we had not used the generational scheme proposed. Another interesting factor to consider is the type of crossover or `Cross_Type` whose means plot can be seen in Fig. 3.

As shown in the graph, the *Similar Block 2-Point Order Crossover* or SB2OX is the best crossover among the nine tested, this supports the idea that maintaining the blocks of jobs in the crossover improves the performance of the algorithm. The remaining means plots for all other factors and experiments are not shown here for reasons of space. After obtaining the best operators and parameters for all 12 experiments we obtain the results shown in Table 2.

The resulting algorithms for all 12 experiments are very much alike. More precisely, the selection and crossover operators and the mutation probability all take the same value: Tournament, SB2OX and 0.01 respectively, and therefore those parameters and operators are used in the final GA_H. The G_r restart scheme parameter proved to be marginally significant in all experiments and

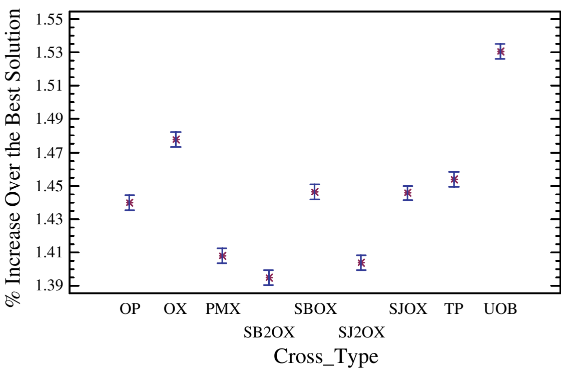

Fig. 3. Means and 95.0% LSD intervals plot for the type of crossover factor (`Crossover_Type`), experiment SSD10_P13.

Table 2
GA_H proposed algorithm calibration results for the 12 different experiments

Experiment	Operators and parameters		
	`Restart`	`Cross_Prob`	`Pop_Size`
SSD10_P13	50	0.1	20
SSD50_P13	50	0.1	20
SSD100_P13	75	0.1	20
SSD125_P13	50	0.1	20
SSD10_P2	50	0.1	50
SSD50_P2	75	0.1	50
SSD100_P2	50	0.2	50
SSD125_P2	50	0.2	50
SSD10_P3	25	0.1	50
SSD50_P3	50	0.1	50
SSD100_P3	50	0.2	50
SSD125_P3	50	0.2	50

`Restart` = Restart operator, `Cross_Prob` = Crossover probability, `Pop_Size` = Population size.

that is why all three possible levels appear in the table. However, the 50 value is the most common and consequently it is chosen. The crossover probability also varies but applying the same logic we choose 0.1. Lastly, the population size is fixed at 50. Although not shown here, we observed a typical "bath-curve" in the means plots of almost all quantitative factors, specially in the crossover and mutation probabilities, meaning that both higher and lower values from the ones indicated in Table 2 yielded worse results. This validates the chosen range of values for these factors in the experiments.

It is interesting to study this unusually low value for the crossover probability which supports the commented tendency for the crossover operators to produce offspring with worse $C_{\max}$ values specially in the latter stages of the algorithm. The experiments included a "validation" value for P_c of 0.0. If the crossover had consistently given bad results this P_c value of 0.0 would have been the best level in the experiment. This was the case for crossovers UOB and OX when we studied the interaction between P_c and the type of crossover. For UOB and OX it is even better not to use crossover at all (P_c of 0.0). SB2OX crossover improved the results of the proposed GA but only when applied at low probabilities. The experiment shown here does not include a validation value for G_r

but previous experiments without restarts (i.e. $G_r = \infty$) showed such poor results that we disregarded the idea of not using restarts at all.

5. Computational evaluation

After having calibrated the GA_H algorithm we aim to test its relative effectiveness and efficiency. The problem is that, as we have seen in Section 3, no general algorithm has been proposed for this problem and therefore, we can not carry out any direct comparison. To overcome this difficulty we have coded several adaptations of many of the metaheuristic methods proposed for the permutation flowshop problem. We have chosen some classic methods as well as other well performing recent metaheuristics (see Ruiz and Maroto, 2005).

The adaptation is carried out by changing the way the $C_{\max}$ is calculated in all methods by the one shown in Section 4.1. The first adapted method has been already cited and is the NEH_H based on the NEH heuristic of Nawaz et al. (1983). The simulated annealing of Osman and Potts (1989) was also changed with this new makespan evaluation. We will refer to this modified method as SA_OP_H. The GA of Reeves (1995) is adapted by changing the initialization from NEH to NEH_H and also by adapting the makespan evaluation. We will call this method $GAReev_H$. We have also changed the tabu search based algorithm of Widmer and Hertz (1989). In this case we also changed the makespan evaluation. This new algorithm will be called $Spirit_H$. Similarly we have adapted the genetic algorithms of Chen et al. (1995), Murata et al. (1996) and Aldowaisan and Allahvedi (2003) that will be referred to as $GAChen_H$, $GAMIT_H$ and GA_AA_H respectively. Recently, Rajendran and Ziegler (2004) published two competitive ant-based algorithms that we have also adapted here for the evaluation. These two algorithms will be referred to as $M\text{-}MMAS_H$ and $PACO_H$ following the author's nomenclature. We also consider a simple RANDOM method that generates and evaluates a given number of solutions and returns the best one as a result. In total we have 10 new methods for solving the real problem, 9 of them being adaptations of published methods.

We carried out 12 different evaluations, one with each instance set with the termination criterion set at the evaluation of 5000 $C_{\max}$. For the tests we use the same computing platform as in Section 4.4. The programming platform used for the coding has been Delphi 7.0. All methods, apart from the NEH_H and $Spirit_H$ are stochastic in nature and we have conducted five different replicates of each experiment to finally average the results. The results of all 12 experiments can be seen in Tables 3–5.

As expected, the RANDOM rule is the worst performer in all 12 experiments. From all other methods we observe a difference between those that are initialized with the NEH_H (GA_H, $GAReev_H$, GA_AA_H, $M\text{-}MMAS_H$ and $PACO_H$) and the others, with the sole exception of the SA_OP_H. Initializing the metaheuristics with a very good starting solution provides very good results on average. From the test it is not clear whether steady state genetic algorithms (GA_H and $GAReev_H$) show a better performance than regular generational GAs ($GAChen_H$, $GAReev_H$, $GAMIT_H$ and GA_AA_H) for being of the steady state type or because the two considered steady state genetic algorithms use NEH_H for initialization. However, GA_AA_H also uses this initialization and the results are consistently worse in all experiments.

An observed trend after the experiments is the tendency towards much worse results as the complexity of the problems increases. The most complex problems are those in the last experiment (SSD125_P3) and we see that all algorithms, with the exception of the GA_H, have average results that are 3% superior to the best solutions obtained. From the results we can conclude that simple metaheuristics like SA_OP_H, $Spirit_H$, $GAChen_H$ or $GAMIT_H$ produce worse results than the other more elaborated and complex methods.

The two most recent methods considered, $M\text{-}MMAS_H$ and $PACO_H$ show a good performance especially in the first experiments and in experiments where the setup times are smaller than the processing times (SSD10 and SSD50). In all three tables we see how both methods deteriorate rapidly as the setup times increase. This results might be motivated by the fact that the algorithms need

Table 3
Average percentage increase over the best known solution for the evaluated methods with the termination criterion set at 5000 makespan evaluations

SSD10_P13 Instances

Instance	RAN-DOM	GA$_H$	SA_OP$_H$	Spirit$_H$	GA Reev$_H$	NEH$_H$	GA Chen$_H$	GA_AA$_H$	GA MIT$_H$	M-MMAS$_H$	PACO$_H$
20 × 5	3.86	0.06	0.70	1.60	0.63	1.78	2.14	1.25	1.31	1.26	0.72
20 × 10	6.07	0.16	1.16	3.19	0.66	2.36	3.05	1.84	2.07	1.80	1.26
20 × 20	7.53	0.22	1.42	3.77	0.80	3.63	3.90	3.32	2.68	3.14	2.75
50 × 5	5.67	0.11	0.71	1.35	0.43	1.71	2.66	1.21	2.34	1.19	0.76
50 × 10	9.21	0.22	2.12	3.11	0.80	2.80	3.74	2.30	4.66	2.24	1.74
50 × 20	10.96	0.33	1.99	3.93	0.96	3.39	4.97	3.01	5.24	2.86	2.46
100 × 5	8.36	0.50	2.29	5.55	0.68	1.97	4.85	1.57	4.65	1.47	1.03
100 × 10	8.42	0.36	2.08	5.91	0.62	1.76	4.83	1.45	4.64	1.13	0.76
100 × 20	9.54	0.37	1.18	6.18	0.53	2.25	4.85	1.64	4.41	1.82	1.07
200 × 10	7.08	0.90	0.82	8.43	0.69	1.39	4.79	0.98	2.66	0.71	0.17
200 × 20	8.72	1.04	1.30	10.35	0.84	1.78	5.05	1.37	3.58	1.22	0.83
Average	7.77	0.39	1.43	4.85	0.70	2.26	4.07	1.81	3.48	1.71	1.23

SSD50_P13 Instances

Instance	RAN-DOM	GA$_H$	SA_OP$_H$	Spirit$_H$	GA Reev$_H$	NEH$_H$	GA Chen$_H$	GA_AA$_H$	GA MIT$_H$	M-MMAS$_H$	PACO$_H$
20 × 5	14.54	0.99	4.22	7.09	2.64	5.77	8.37	5.01	6.34	5.06	5.01
20 × 10	12.40	0.16	3.42	6.39	2.32	6.13	6.25	5.35	5.43	5.72	5.23
20 × 20	10.90	0.63	3.43	6.05	2.62	6.17	5.53	5.11	4.84	5.59	5.09
50 × 5	20.82	0.71	3.32	6.04	2.09	4.80	10.49	3.90	9.41	4.28	3.74
50 × 10	18.20	1.01	3.92	6.50	2.67	5.05	8.59	4.24	9.40	4.66	3.92
50 × 20	16.01	0.66	3.66	5.79	2.12	4.50	7.65	3.69	8.50	4.01	3.55
100 × 5	23.24	1.48	4.47	16.75	1.54	3.78	11.86	2.90	12.04	3.16	2.79
100 × 10	20.85	1.27	3.90	14.98	1.86	3.44	10.69	2.71	10.79	3.00	2.41
100 × 20	19.26	1.68	3.16	13.74	2.16	4.07	10.27	3.41	9.89	3.53	3.38
200 × 10	24.42	2.55	3.70	27.46	2.16	3.03	14.98	2.18	10.09	2.55	2.12
200 × 20	18.61	2.18	3.30	20.88	1.91	2.65	10.45	1.85	8.60	2.35	1.60
Average	18.11	1.21	3.68	11.97	2.19	4.49	9.56	3.67	8.67	3.99	3.53

SSD100_P13 Instances

Instance	RAN-DOM	GA$_H$	SA_OP$_H$	Spirit$_H$	GA Reev$_H$	NEH$_H$	GA Chen$_H$	GA_AA$_H$	GA MIT$_H$	M-MMAS$_H$	PACO$_H$
20 × 5	22.58	1.25	6.59	11.56	5.44	9.86	14.58	9.38	10.07	9.27	9.34
20 × 10	17.07	1.13	7.04	10.21	4.40	8.54	10.35	8.10	8.63	8.08	7.70
20 × 20	13.88	1.19	4.88	8.10	3.83	8.28	8.26	7.83	6.94	7.86	7.76
50 × 5	34.83	1.35	6.15	9.47	4.12	7.58	18.87	7.10	16.42	6.98	6.87
50 × 10	25.00	0.55	5.40	7.75	3.00	7.23	12.45	6.78	12.16	6.67	6.74
50 × 20	19.87	0.57	4.58	6.85	2.79	5.81	10.40	5.37	10.59	5.18	5.24
100 × 5	37.88	1.87	6.19	27.21	2.26	4.67	21.51	4.26	19.09	4.18	4.16
100 × 10	31.25	2.01	5.35	23.03	2.50	4.70	17.54	4.26	15.65	4.37	4.06
100 × 20	26.50	2.07	4.19	19.19	2.47	4.83	15.23	4.46	13.85	4.40	4.55
200 × 10	40.47	3.22	6.55	46.17	2.74	3.67	26.61	3.12	17.42	3.16	3.11
200 × 20	27.76	2.97	5.64	30.77	2.55	3.40	17.10	3.03	13.17	2.93	2.73
Average	27.01	1.65	5.69	18.21	3.28	6.23	15.72	5.79	13.09	5.73	5.66

SSD125_P13 Instances

Instance	RAN-DOM	GA$_H$	SA_OP$_H$	Spirit$_H$	GA Reev$_H$	NEH$_H$	GA Chen$_H$	GA_AA$_H$	GA MIT$_H$	M-MMAS$_H$	PACO$_H$
20 × 5	27.08	1.80	8.17	13.93	6.10	11.52	15.95	9.09	12.03	11.20	10.29
20 × 10	19.37	1.56	7.86	10.82	4.85	9.80	11.77	8.11	9.92	9.59	8.82
20 × 20	14.36	0.64	6.25	8.38	3.61	7.51	8.63	5.72	6.99	7.02	6.31
50 × 5	39.15	1.36	6.86	10.78	3.75	9.15	21.02	9.01	18.48	8.83	8.40
50 × 10	27.66	0.14	5.28	8.50	3.61	7.57	14.94	7.44	14.08	7.07	6.77
50 × 20	22.38	0.64	5.49	8.25	3.44	6.91	11.97	6.75	11.94	6.48	6.19
100 × 5	43.65	2.01	7.22	31.51	2.62	5.45	24.57	5.45	22.14	5.03	4.49
100 × 10	34.97	2.14	5.54	25.36	2.68	5.23	20.48	5.23	17.57	4.85	4.09
100 × 20	29.16	1.91	4.20	21.36	2.54	4.78	16.56	4.78	15.11	4.45	3.93
200 × 10	46.67	3.32	7.42	52.02	2.81	3.81	30.14	3.81	19.77	3.28	3.18
200 × 20	31.73	2.90	6.89	34.57	2.57	3.46	20.12	3.46	15.04	3.01	2.63
Average	30.56	1.68	6.47	20.50	3.51	6.83	17.83	6.26	14.82	6.44	5.92

Experiments SSD10_P13, SSD50_P13, SSD100_P13 and SSD125_P13.

Table 4
Average percentage increase over the best known solution for the evaluated methods with the termination criterion set at 5000 makespan evaluations

SSD10_P2 Instances

Instance	RAN-DOM	GA$_H$	SA_ OP$_H$	Spirit$_H$	GA Reev$_H$	NEH$_H$	GA Chen$_H$	GA_ AA$_H$	GAMIT$_H$	M-MMAS$_H$	PACO$_H$
20 × 5	13.46	2.64	8.64	12.54	3.77	9.94	7.55	5.91	7.86	9.41	9.26
20 × 10	11.83	1.76	5.99	9.52	4.12	7.67	6.57	5.29	7.34	7.12	7.07
20 × 20	9.22	1.23	4.51	6.73	3.12	5.74	5.29	3.71	5.48	5.38	5.24
50 × 5	13.37	1.43	8.63	7.86	1.90	4.61	4.94	4.04	9.53	4.06	4.14
50 × 10	13.97	1.35	8.46	9.01	2.58	4.61	5.42	3.99	9.61	4.02	4.16
50 × 20	12.40	0.35	7.24	8.03	1.85	3.25	4.22	2.59	8.39	2.92	2.75
100 × 5	11.60	0.95	7.73	7.65	1.35	2.64	4.54	2.20	9.03	2.25	2.21
100 × 10	11.64	0.94	7.79	8.45	1.55	2.62	2.84	2.31	9.07	2.24	2.20
100 × 20	11.37	0.79	7.48	8.62	1.52	2.61	3.01	2.11	8.31	2.15	2.01
200 × 10	8.19	0.65	6.75	10.56	0.63	0.99	2.15	0.48	6.82	0.45	0.48
200 × 20	8.61	0.59	6.41	10.88	0.61	0.84	2.15	0.30	6.92	0.25	0.34
Average	11.42	1.15	7.24	9.08	2.09	4.14	4.43	2.99	8.03	3.66	3.62

SSD50_P2 Instances

Instance	RAN-DOM	GA$_H$	SA_ OP$_H$	Spirit$_H$	GA Reev$_H$	NEH$_H$	GA Chen$_H$	GA_ AA$_H$	GA MIT$_H$	M-MMAS$_H$	PACO$_H$
20 × 5	14.35	3.10	11.70	12.79	6.33	13.15	9.08	7.46	9.75	12.70	12.62
20 × 10	11.45	1.38	8.00	10.78	4.55	9.30	6.57	5.32	7.23	8.69	8.44
20 × 20	8.84	1.13	5.70	7.07	2.73	5.91	4.99	3.50	5.65	5.40	5.39
50 × 5	14.08	1.74	10.35	9.66	3.64	6.33	4.77	5.77	10.11	5.86	5.48
50 × 10	12.35	1.76	8.52	8.94	3.04	5.46	4.47	4.81	9.08	4.84	4.82
50 × 20	9.95	1.15	6.50	7.08	2.77	4.40	3.11	3.82	6.83	3.91	3.85
100 × 5	12.07	0.82	8.02	8.05	1.52	2.54	5.43	2.17	9.60	1.92	1.80
100 × 10	10.17	0.98	7.01	7.73	1.52	2.46	2.99	2.11	8.09	2.07	1.88
100 × 20	8.62	0.53	5.90	6.54	1.37	2.14	1.82	1.70	6.78	1.70	1.48
200 × 10	7.62	0.77	5.90	9.14	0.81	1.09	3.63	0.62	6.70	0.50	0.54
200 × 20	6.19	0.57	4.79	7.20	0.63	0.94	2.17	0.58	5.51	0.35	0.50
Average	10.52	1.27	7.49	8.63	2.63	4.88	4.46	3.44	7.76	4.36	4.25

SSD100_P2 Instances

Instance	RAN-DOM	GA$_H$	SA_ OP$_H$	Spirit$_H$	GA Reev$_H$	NEH$_H$	GA Chen$_H$	GA_ AA$_H$	GAMIT$_H$	M-MMAS$_H$	PACO$_H$
20 × 5	15.58	4.09	13.85	15.31	7.59	14.72	11.12	9.17	11.80	13.66	13.91
20 × 10	11.71	2.40	9.81	11.25	5.66	9.42	6.93	6.31	8.28	8.84	8.61
20 × 20	9.35	1.64	7.27	8.07	4.09	7.48	5.30	4.79	6.26	6.93	6.83
50 × 5	14.96	2.15	10.49	9.62	4.16	7.82	5.94	7.28	10.88	7.34	7.20
50 × 10	11.94	1.35	8.44	9.53	3.98	5.77	4.65	5.00	8.91	5.21	5.21
50 × 20	9.76	1.10	6.70	7.05	3.34	5.29	3.53	4.55	7.20	4.77	4.61
100 × 5	12.67	1.14	8.35	8.97	1.91	3.06	5.51	2.63	9.87	2.53	2.43
100 × 10	9.58	0.76	6.29	7.60	1.99	3.18	4.01	2.77	7.69	2.40	2.57
100 × 20	8.32	0.74	5.70	6.62	1.85	2.61	2.95	2.21	6.74	2.08	1.99
200 × 10	8.19	1.01	6.38	9.00	1.25	1.56	5.76	1.17	7.35	0.96	1.11
200 × 20	6.22	0.97	4.98	6.67	1.18	1.49	4.07	0.89	5.43	0.95	0.86
Average	10.75	1.58	8.02	9.06	3.36	5.67	5.43	4.25	8.22	5.06	5.03

SSD125_P2 Instances

Instance	RAN-DOM	GA$_H$	SA_ OP$_H$	Spirit$_H$	GA Reev$_H$	NEH$_H$	GA Chen$_H$	GA_ AA$_H$	GA MIT$_H$	M-MMAS$_H$	PACO$_H$
20 × 5	16.60	3.96	15.09	16.54	9.07	17.45	12.24	11.01	12.43	16.82	16.72
20 × 10	12.28	2.07	11.23	13.07	5.88	10.03	8.33	6.43	9.21	9.42	9.46
20 × 20	8.85	1.46	6.61	8.55	4.18	6.88	5.58	4.42	5.84	6.58	6.14
50 × 5	15.39	3.13	10.75	10.96	5.38	7.84	6.72	7.19	11.44	7.40	7.23
50 × 10	11.98	1.32	9.40	9.25	4.78	7.07	4.85	6.31	9.06	6.52	6.25
50 × 20	9.35	1.16	6.63	7.03	3.14	5.64	3.93	4.80	7.01	5.13	5.09
100 × 5	13.58	1.06	8.99	9.67	2.11	3.42	6.29	2.92	10.66	2.82	2.79
100 × 10	10.04	0.58	6.51	7.64	1.61	2.43	4.44	1.95	7.93	1.94	1.75
100 × 20	7.24	0.67	5.04	5.72	1.86	2.58	2.73	2.02	5.78	1.92	1.90
200 × 10	8.59	0.97	6.53	9.42	1.22	1.61	6.42	1.19	7.56	0.91	1.17
200 × 20	6.14	0.84	4.85	6.95	1.10	1.45	4.33	1.19	5.45	0.90	0.91
Average	10.91	1.56	8.33	9.53	3.67	6.04	5.99	4.49	8.40	5.49	5.40

Experiments SSD10_P2, SSD50_P2, SSD100_P2 and SSD125_P2.

Table 5
Average percentage increase over the best known solution for the evaluated methods with the termination criterion set at 5000 makespan evaluations

SSD10_P3 Instances

Instance	RAN-DOM	GA$_H$	SA_OP$_H$	Spirit$_H$	GA Reev$_H$	NEH$_H$	GA Chen$_H$	GA_AA$_H$	GA MIT$_H$	M-MMAS$_H$	PACO$_H$
20 × 5	13.01	3.06	9.53	13.33	4.01	11.28	8.40	6.34	9.22	10.65	10.60
20 × 10	11.74	2.34	6.39	11.79	3.25	6.81	7.11	4.83	7.34	6.25	6.14
20 × 20	8.82	1.03	4.51	7.71	2.33	4.84	5.56	3.56	5.24	4.18	4.15
50 × 5	14.03	2.33	10.42	10.89	3.05	7.16	4.21	6.12	10.78	6.49	6.88
50 × 10	14.22	1.96	12.22	11.67	2.62	5.23	4.81	4.70	10.61	4.65	4.64
50 × 20	13.14	0.87	8.82	11.04	2.43	3.85	5.17	3.22	9.93	3.31	3.35
100 × 5	11.48	1.48	10.18	10.19	2.06	3.23	4.59	2.94	10.26	2.72	2.77
100 × 10	11.30	0.54	9.46	10.14	0.97	2.32	2.54	1.88	9.83	1.70	1.78
100 × 20	10.93	0.41	9.15	10.16	0.97	1.97	2.72	1.45	9.38	1.45	1.45
200 × 10	9.65	0.60	9.38	12.51	0.62	0.99	4.27	0.54	9.62	0.38	0.18
200 × 20	7.37	0.32	6.86	10.05	0.57	0.97	0.95	0.51	7.27	0.45	0.49
Average	11.43	1.36	8.81	10.86	2.08	4.42	4.58	3.28	9.04	3.84	3. 86

SSD50_P3 Instances

Instance	RAN-DOM	GA$_H$	SA_OP$_H$	Spirit$_H$	GA Reev$_H$	NEH$_H$	GA Chen$_H$	GA_AA$_H$	GA MIT$_H$	M-MMAS$_H$	PACO$_H$
20 × 5	13.89	4.22	11.35	15.49	6.73	15.16	10.27	8.45	10.68	14.78	14.40
20 × 10	12.42	2.86	9.90	12.70	5.18	10.67	8.26	6.57	8.99	9.98	9.81
20 × 20	9.63	1.13	7.51	8.87	3.39	5.27	6.21	3.91	6.53	4.66	4.59
50 × 5	14.06	2.50	11.33	10.94	3.35	7.34	5.56	6.39	10.75	6.82	6.70
50 × 10	12.00	2.09	11.07	10.30	3.72	6.48	4.23	5.49	9.55	6.00	6.09
50 × 20	10.23	1.11	8.41	9.27	2.72	4.93	3.36	4.27	8.07	4.41	4.16
100 × 5	10.51	1.17	8.40	7.96	1.91	3.42	6.84	3.08	9.47	2.68	2.92
100 × 10	8.59	0.88	7.06	7.34	1.87	2.86	3.27	2.35	7.88	2.25	2.20
100 × 20	7.69	0.45	6.57	7.24	1.06	1.70	1.95	1.16	6.68	1.20	1.19
200 × 10	6.72	0.63	5.76	8.22	0.69	1.06	4.27	0.57	6.85	0.44	0.52
200 × 20	5.48	0.55	4.96	6.82	0.95	1.32	2.61	0.80	5.37	0.75	0.77
Average	10.11	1.60	8.39	9.56	2.87	5.47	5.16	3.91	8.26	4.91	4.85

SSD100_P3 Instances

Instance	RAN-DOM	GA$_H$	SA_OP$_H$	Spirit$_H$	GA Reev$_H$	NEH$_H$	GA Chen$_H$	GA_AA$_H$	GA MIT$_H$	M-MMAS$_H$	PACO$_H$
20 × 5	15.62	5.89	15.63	17.11	8.60	17.39	11.33	11.44	12.46	17.39	17.39
20 × 10	12.05	2.20	10.54	12.46	5.38	12.29	8.34	6.62	9.53	12.29	12.21
20 × 20	10.09	1.81	9.48	9.39	4.62	8.34	6.44	5.74	7.17	8.34	8.34
50 × 5	13.56	3.15	10.83	11.41	4.55	8.91	6.41	8.33	11.12	8.91	8.91
50 × 10	12.14	2.91	11.85	11.14	4.52	6.29	5.47	6.20	10.03	6.29	6.29
50 × 20	9.15	1.32	7.66	8.13	3.24	6.08	2.92	5.36	7.36	6.08	6.08
100 × 5	11.17	1.02	9.05	8.78	2.05	3.10	10.68	3.10	9.63	3.10	3.10
100 × 10	7.88	0.73	6.01	6.04	1.44	2.52	5.55	2.52	7.28	2.52	2.52
100 × 20	6.18	0.76	4.86	5.66	1.44	2.68	3.24	2.68	5.49	2.68	2.68
200 × 10	6.61	0.58	5.34	7.44	0.68	1.02	6.60	1.02	6.40	1.02	1.02
200 × 20	4.71	0.51	4.21	5.45	1.04	1.48	3.98	1.48	4.68	1.48	1.48
Average	9.92	1.90	8.68	9.36	3.42	6.37	6.45	4.95	8.29	6.37	6. 36

SSD125_P3 Instances

Instance	RAN-DOM	GA$_H$	SA_OP$_H$	Spirit$_H$	GA Reev$_H$	NEH$_H$	GA Chen$_H$	GA_AA$_H$	GA MIT$_H$	M-MMAS$_H$	PACO$_H$
20 × 5	16.58	5.69	16.40	18.86	8.97	19.47	12.01	11.66	13.12	18.92	18.40
20 × 10	12.30	3.67	12.01	13.70	7.99	15.72	8.53	8.30	10.10	15.12	15.05
20 × 20	9.31	1.65	8.53	9.77	4.60	7.60	6.34	4.70	6.89	7.20	6.89
50 × 5	15.34	3.66	12.51	12.86	5.25	9.07	8.75	8.34	12.27	8.47	8.45
50 × 10	12.36	3.66	11.30	11.12	5.53	9.03	5.80	8.01	10.40	8.37	8.52
50 × 20	8.73	1.42	7.29	8.34	3.34	5.35	2.89	4.58	7.58	4.74	4.67
100 × 5	11.34	0.92	8.90	8.62	2.02	3.59	11.21	3.20	9.82	3.23	3.12
100 × 10	8.55	0.73	6.89	6.70	1.96	2.98	7.40	2.40	7.84	2.52	2.53
100 × 20	5.77	0.30	4.37	5.27	1.52	2.42	3.84	1.88	5.08	1.88	1.90
200 × 10	5.49	0.60	4.31	6.33	0.79	1.39	6.19	0.80	5.09	0.96	0.87
200 × 20	4.92	0.73	4.08	5.57	1.30	1.71	4.92	1.21	4.79	1.21	1.30
Average	10.06	2.09	8.78	9.74	3.93	7.12	7.08	5.01	8.45	6.60	6.52

Experiments SSD10_P3, SSD50_P3, SSD100_P3 and SSD125_P3.

to be reworked in order to consider setup times explicitly when calculating and updating the pheromone matrices. Nevertheless, both methods show a good performance in the problems of 200 jobs and 10 or 20 machines in the first two experiments. A closer observation along with the results of the NEH_H shows that both ant algorithms are able to significantly improve the schedules obtained by NEH_H. This outcome is motivated by the fact that both ant algorithms incorporate an efficient form of local search.

The two best methods in almost all tests are the $GAReev_H$ and our proposed algorithm GA_H. Furthermore, the average performance of this new genetic algorithm is superior to all the others in all tests. In general terms, the proposed GA_H improves the $GAReev_H$ in all the experiments by a margin of between 53% and 135% depending on the case. The reasons behind these results might be that both GA algorithms are able to explore wider areas in the search space under the stopping criterion considered here. The addition of local search to the $M\text{-}MMAS_H$ and $PACO_H$ methods shows good results in the easiest experiments but this local search is expensive in terms of objective function evaluations, thus leaving parts of the search space to explore.

The above results are further confirmed by an analysis of variance. In Fig. 4 we show a means plot for the type of algorithm in the experiment SSD10_P13.

We can see that the tested methods are statistically different. The algorithm GA_H is statistically

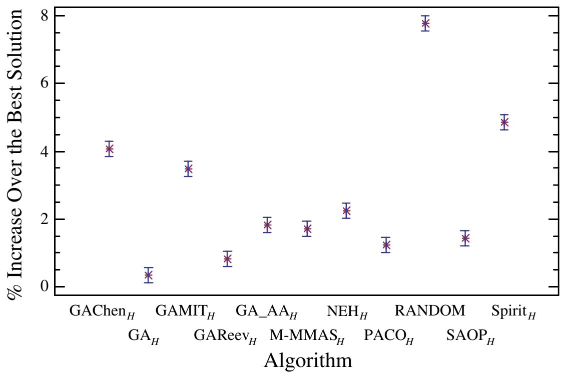

Fig. 4. Means and 95.0% LSD intervals plot for the type of algorithm, experiment SSD10_P13.

better than $GAReev_H$. Although not shown here, the results for all the other 11 experiments are better since the experiment SSD10_P13 is the one where the gap between GA_H and $GAReev_H$ is the smallest.

Regarding the CPU time needed by the different methods we can say that there is almost no difference. In Table 6 we show the CPU times needed for all methods in the SSD10_P13 experiment.

We can see that all methods needed between 22 and 25 seconds on average to evaluate 5000 $C_{\max}$, with the exception of the NEH_H heuristic which evaluates less makespans and therefore is much faster. Although the times do increase in the instance sets with two and three machines per stage, there is almost no difference in CPU times between all metaheuristics. The reason behind this result is that the calculation of the $C_{\max}$ takes up a large part of the times a given method needs, and since all metaheuristics calculate the same number of makespans, the times are similar. Thus, we can now safely say that our proposed method GA_H is the most effective algorithm for the problem under consideration.

Apart from solving random instances, we have carried out an additional experiment with a set of real data taken from ceramic tile companies. More precisely we have obtained a set of 10 real production problems where we know all the relevant data: processing times, setup times and eligibility constraints. The real problems are taken from a company of medium size that has three production lines where we only find two production stages, that is, the total number of machines ($\sum_{i=1}^{m} m_i$) is 6. For each one of these 10 problems we know the real programming that was done by the production manager at the plant. In this situation we can compare the $C_{\max}$ obtained manually and the one that can be obtained with the proposed algorithm GA_H. Obtaining all the necessary data has been a really complicated and lengthy task. The company has more than 300 different products and six machines. Subsequently, we gathered a matrix of size 300×6 for the processing times and six matrices of 300×300 for the setup times. All 10 problems have been solved with the GA_H algorithm with the stopping criterion set at the evaluation of 50,000 $C_{\max}$. We have carried

Table 6
CPU times (in seconds) for the different evaluated methods—Experiment SSD10_P3

Instance	RANDOM	GA_H	SA_OP_H	$Spirit_H$	$GAReev_H$	NEH_H	GA $Chen_H$	GA_ AA_H	GA MIT_H	$M\text{-}MMAS_H$	$PACO_H$
20 × 5	0.90	2.19	0.94	0.81	2.10	<0.5	1.14	1.61	0.87	2.16	1.59
20 × 10	1.43	2.83	1.58	1.46	2.75	<0.5	1.72	2.23	1.52	2.80	2.38
20 × 20	2.95	4.37	3.13	2.99	4.27	<0.5	3.23	3.73	3.05	4.34	3.66
50 × 5	1.82	3.13	1.94	1.79	3.85	<0.5	2.48	3.15	1.86	3.51	3.20
50 × 10	4.16	5.45	4.40	4.17	6.22	<0.5	4.84	5.51	4.27	5.86	5.89
50 × 20	11.17	10.14	9.12	8.69	10.73	<0.5	9.79	10.19	8.88	10.48	9.67
100 × 5	5.63	5.88	4.58	4.33	8.42	<0.5	6.79	7.58	4.44	7.18	7.18
100 × 10	17.61	11.74	9.86	9.72	13.86	0.63	16.51	15.12	9.74	12.86	16.05
100 × 20	47.32	47.61	45.75	46.66	49.70	2.32	49.84	49.56	46.04	48.91	48.11
200 × 10	51.34	43.22	48.88	50.39	49.91	10.11	57.40	53.41	49.43	46.85	57.22
200 × 20	114.89	102.71	114.58	117.09	109.92	30.15	122.10	115.38	115.32	106.66	112.20
Average	23.56	21.75	22.25	22.55	23.79	3.98	25.08	24.31	22.31	22.87	24.29

Table 7
Results of the proposed genetic algorithm and manual programming for 10 real production programming examples

Instance	n	m	M	Average $C_{\max}$ for the GA_H algorithm	"Manual" $C_{\max}$	Difference (%)
TEST1P	32	2	6	22943.1	24752	7.88
TEST2P	32	2	6	23252.2	25061	7.78
TEST3P	32	2	6	23245.4	25089	7.93
TEST4P	32	2	6	23937.1	25842	7.96
TEST5P	32	2	6	24339.2	24904	2.32
TEST6P	32	2	6	24719.1	25330	2.47
TEST7P	32	2	6	25929.8	28135	8.50
TEST8P	32	2	6	26784.1	30302	13.13
TEST9P	32	2	6	27889.5	31807	14.05
TEST10P	32	2	6	27780.5	32490	16.95
					Average	8.90

out five independent runs and the results have been averaged out. The result is compared to the "manual" $C_{\max}$ obtained at the plant. The data can be seen in Table 7.

All 10 problems considered were comprised of 32 independent jobs (a common size for production programming at the company) and from the results we can see that the GA_H algorithm can obtain real, feasible schedules that are between 2.32% and 16.95% better than the manual schedules. Overall, the decrease in the $C_{\max}$ is almost 9%. This result can be considered as very promising since the company can increase its production capacity by nearly 9% by just using the algorithm in the weekly production programming. Additionally, the problems tested are small compared to the previously evaluated random instances and the running times for the GA_H algorithm were of 2.27 seconds on average, which is several orders of magnitude faster than the time it takes to develop a manual schedule.

This algorithm has been coded inside a complete production programming tool called ProdPlanner that is currently being employed at several important companies in the ceramic tile sector of Spain. Initial results show that as the real problems become larger (more jobs, stages and machines per stage), the differences between the manual methods and the proposed GA_H algorithm widen considerably.

6. Conclusions

In this study we have proposed a new genetic algorithm for a complex scheduling problem, which can be viewed as a generalization of other simpler problems like the regular permutation flowshop, or the flowshop with multiple processors (FSMP). More precisely, we have shown a new heuristic based on a genetic algorithm that solves the hybrid flowshop with sequence dependent setup times, unrelated parallel machines at each stage and machine eligibility constraints. This problem is common in the ceramic tile industries, as well as in the fabric manufacturing sector. The proposed genetic algorithm includes several procedures such as the restart and the generational schemes, as well as four new crossover operators. The algorithm has been carefully calibrated by means of extensive experiments using a comprehensive set of 1320 problems. Also, we have coded nine adaptations of other metaheuristics that have shown good performance in simpler production programming environments. A computational experiment has been conducted to check the performance of the proposed algorithm. The statistically significant results indicate that the proposed algorithm is between 53% and 135% better than the second best method. We have also tested the algorithm in a situation with real data and the results show a decrease in the makespan of the schedules generated by the algorithm with respect to manual methods of almost 9% in simple problems.

We are currently working on installing this algorithm in several important firms of the Spanish ceramic tile sector and on considering other constraints such as machine-based time lags, limited storage capacity between stages and production priorities. These extensions are to bridge the existing gap between the theory of scheduling and its applications in real industrial settings.

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

Acknowledgments

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 like to thank D. José Romeo from the company Cerypsa Cerámicas, S.A. and to D. Javier Marqués from the company Halcón Cerámicas S.A. for their support and collaboration in this work. Finally, our thanks to the four anonymous referees whose comments have considerably improved an earlier version of this paper.

References

Adler, L., Fraiman, N., Kobacker, E., Pinedo, M., Plotnicoff, T.P., Wu, T.P., 1993. BPSS: A scheduling support system for the packaging industry. Operations Research 41 (4), 641–648.

Aghezzaf, E.H., Artiba, A., Moursli, O., Tahon, C., 1995. Hybrid flowshop problems, a decomposition based heuristic approach. In: Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'95, Marrakech. FUCAM-INRIA, pp. 43–56.

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, 614–626.

Aldowaisan, T., Allahvedi, A., 2003. New heuristics for no-wait flowshops to minimize makespan. Computers & 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.

Andrés, C., 2001. Programación de la Producción en Talleres de Flujo Híbridos con Tiempos de Cambio de Partida Dependientes de la Secuencia. Modelos, Métodos y Algoritmos de Resolución. Aplicación a Empresas del Sector Cerámico. PhD thesis, Departamento de Organización de Empresas. Universidad Politécnica de Valencia (in Spanish).

Arthanary, T.S., Ramaswamy, K.G., 1971. An extension of two machine sequencing problem. OPSEARCH, The Journal of the Operational Research Society of India 8 (4), 10–22.

Brah, S.A., Hunsucker, J.L., 1991. Branch and bound algorithm for the flow shop with multiple processors. European Journal of Operational Research 51, 88–99.

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.

Dudek, R.A., Panwalkar, S.S., Smith, M.L., 1992. The lessons of flowshop scheduling research. Operations Research 40 (1), 7–13.

Ford, F.N., Bradbard, D.A., Ledbetter, W.N., Cox, J.F., 1987. Use of operations research in production management. Production and Inventory Management 28 (3), 59–62.

Gourgand, M., Grangeon, N., Norre, S., 1999. Metaheuristics for the deterministic hybrid flow shop problem. In: Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'99, Glasgow. FUCAM-INRIA, pp. 136–145.

Graves, S.C., 1981. A review of production scheduling. Operations Research 29 (4), 646–675.

Guinet, A.G., 1991. Textile production systems: A succession of non-identical parallel processor shops. Journal of the Operational Research Society 42 (8), 655–671.

Guinet, A.G., Solomon, M.M., 1996. Scheduling hybrid flowshops to minimize maximum tardiness or maximum completion time. International Journal of Production Research 34 (6), 1643–1654.

Gupta, J.N.D., 1988. Two-stage, hybrid flowshop scheduling problem. Journal of the Operational Research Society 39 (4), 359–364.

Gupta, J.N.D., Hariri, A.M.A., Potts, C.N., 1997. Scheduling a two-stage hybrid flow shop with parallel machines at the first stage. Annals of Operations Research 69, 171–191.

Gupta, J.N.D., Tunc, E.A., 1991. Schedules for a two-stage hybrid flowshop with parallel machines at the second stage. International Journal of Production Research 29 (7), 1489–1502.

Johnson, S.M., 1954. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1, 61–68.

Kurz, M.E., Askin, R.G., 2003. Comparing scheduling rules for flexible flow lines. International Journal of Production Economics 85, 371–388.

Kurz, M.E., Askin, R.G., in press. Scheduling flexible flow lines with sequence-dependent setup times. European Journal of Operational Research.

Ledbetter, W.N., Cox, J.F., 1977. Operations research in production management: An investigation of past and present utilization. Production and Inventory Management 18 (3), 84–91.

Leon, V.J., Ramamoorthy, B., 1997. An adaptable problem-space-based search method for flexible flow line scheduling. IIE Transactions 29, 115–125.

Linn, R., Zhang, W., 1999. Hybrid flow shop scheduling: A survey. Computers and Industrial Engineering 37, 57–61.

MacCarthy, B.L., Liu, J., 1993. Addressing the gap in scheduling research: A review of optimization and heuristic methods in production scheduling. International Journal of Production Research 31 (1), 59–79.

McKay, K.N., Pinedo, M., Webster, S., 2002. Practice-focused research issues for scheduling systems. Production and Operations Management 11 (2), 249–258.

McKay, K.N., Safayeni, F.R., Buzacott, J.A., 1988. Job-shop scheduling theory: What is relevant?. Interfaces 4 (18), 84–90.

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

Montgomery, D.C., 2000. Design and Analysis of Experiments, fifth ed. John Wiley.

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.

Nowicki, E., Smutnicki, C., 1998. The flow shop with parallel machines: A tabu search approach. European Journal of Operational Research 106, 226–253.

Olhager, J., Rapp, B., 1995. Operations research techniques in manufacturing planning and control systems. International Transactions in Operational Research 2 (1), 29–43.

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.

Portmann, M.C., Vignier, A., Dardilhac, D., Dezalay, D., 1998. Branch and bound crossed with GA to solve hybrid flowshops. European Journal of Operational Research 107, 389–400.

Rajendran, C., Chaudhuri, D., 1992a. A multi-stage parallel-processor flowshop problem with minimum flowtime. European Journal of Operational Research 57, 111–122.

Rajendran, C., Chaudhuri, D., 1992b. Scheduling in *n*-jobs, *m*-stage flowshops with parallel processors to minimize makespan. International Journal of Production Economics 27, 137–143.

Rajendran, C., Ziegler, H., 2004. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research 155, 426–438.

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

Riane, F., Artiba, A., Elmaghraby, S.E., 1998. A hybrid three-stage flowshop problem: Efficient heuristics to minimize makespan. European Journal of Operational Research 109, 321–329.

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

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

Ruiz, R., Maroto, C., Alcaraz, J., in press. Two new robust genetic algorithms for the flowshop scheduling problem. OMEGA, the International Journal of Management Science.

Salvador, M.S., 1973. A solution to a special case of flow shop scheduling problems. In: Elmaghraby, S.E. (Ed.), Symposium of the Theory of Scheduling and Applications, Springer-Verlag, New York, pp. 83–91.

Santos, D.L., Hunsucker, J.L., Deal, D.E., 1996. An evaluation of sequencing heuristics in flow shops with multiple processors. Computers and Industrial Engineering 30 (4), 681–692.

Sherali, H.D., Sarin, S.C., Kodialam, M.S., 1990. Models and algorithms for a two-stage production process. Production Planning and Control 1 (1), 27–39.

Soewnadi, H., Elmaghraby, S.E., 2001. Sequencing three-stage flexible flowshops with identical machines to minimize makespan. IIE Transactions 31, 985–993.

Sriskandarajah, C., Sethi, S.P., 1989. Scheduling algorithms for flexible flowshops: Worst and average case performance. European Journal of Operational Research 43, 143–160.

Vallada, E., Maroto, C., Ruiz, R., Segura, B., 2005. Análisis de la Programación de la Producción en el Sector Azulejero. Boletín de la Sociedad Española de Cerámica y Vidrio 44 (1), 39–44 (in Spanish).

Vignier, A., Billaut, J.C., Proust, C., 1999. Les Problèmes D'Ordonnancement de Type Flow-Shop Hybride: État de L'Art. RAIRO Recherche opérationnelle 33 (2), 117–183, in French.

Voss, S., 1993. The two-stage hybrid flowshop scheduling problem with sequence-dependent setup times. In: Fandel, T., Gulledge, T., Jones, A. (Eds.), Operations Research in Production Planning and Control. Springer, Berlin, pp. 336–352.

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

Wittrock, R.J., 1985. Scheduling algorithms for flexible flow lines. IBM Journal of Research and Development 29 (4), 401–412.