Available online at www.sciencedirect.com
SCIENCE DIRECT®
Omega 34 (2006) 461–476
omega
The International Journal of Management Science
www.elsevier.com/locate/omega

Two new robust genetic algorithms for the flowshop scheduling problem

Rubén Ruiz*, Concepción Maroto, Javier Alcaraz

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

Received 5 May 2003; accepted 3 December 2004
Available online 9 April 2005

Abstract

The flowshop scheduling problem (FSP) has been widely studied in the literature and many techniques for its solution have been proposed. Some authors have concluded that genetic algorithms are not suitable for this hard, combinatorial problem unless hybridization is used. This work proposes new genetic algorithms for solving the permutation FSP that prove to be competitive when compared to many other well known algorithms. The optimization criterion considered is the minimization of the total completion time or makespan ($C_{\max}$). We show a robust genetic algorithm and a fast hybrid implementation. These algorithms use new genetic operators, advanced techniques like hybridization with local search and an efficient population initialization as well as a new generational scheme. A complete evaluation of the different parameters and operators of the algorithms by means of a Design of Experiments approach is also given. The algorithm's effectiveness is compared against 11 other methods, including genetic algorithms, tabu search, simulated annealing and other advanced and recent techniques. For the evaluations we use Taillard's well known standard benchmark. The results show that the proposed algorithms are very effective and at the same time are easy to implement.

Keywords: Flowshop; Genetic algorithms; Local search

1. Introduction

In a flowshop scheduling problem (FSP) we have n independent jobs $\{J_1, \ldots, J_n\}$ that have to be processed on m different machines $\{M_1, \ldots, M_m\}$. Every job is composed of m operations, and every operation requires a different machine. O_{ij} denotes the operation on machine i of job j. Every operation requires a processing time p_{ij}. In a flowshop all jobs have the same processing order on machines since there is a technological order on the machines for the different jobs to follow. The objective is to find an ordering of the jobs on the machines or *sequence* that optimizes some given criterion. By far, the most common criterion is the minimization of the total completion time of the schedule, often referred to as makespan ($C_{\max}$). There are some other interesting variations to this problem (see [1] or [2]).

It is important to note that there are several additional conditions to this problem [1]:

- All operations are independent and available for processing at time 0.
- All m machines are continuously available.
- Each machine i can process at most one job j at a time.
- Each job j can be processed only on one machine i at a time.

* Corresponding author. Universidad Politécnica de Valencia, Departamento de Estadística e Investigación Operativa Aplicadas y Calidad, Camino de Vera S/N, 46021, Valencia, Spain. Tel.: +34 96 387 70 07x74946; fax: +34 96 387 74 99.
E-mail address: rruiz@eio.upv.es (R. Ruiz).

0305-0483/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.omega.2004.12.006

- The processing of a given operation O_{ij} cannot be interrupted, that is, no preemption is allowed.
- Setup and removal times are sequence independent and are included in the processing times or otherwise are negligible and can be ignored.
- In-process inventory is allowed. If a given operation needs an unavailable machine then the operation joins a queue of unlimited size at that machine.

Following the four parameter notation $A/B/C/D$ of Conway et al. [3] the problem is classified as $n/m/F/F_{\max}$. A more recent three parameter notation $(\alpha/\beta/\gamma)$ was proposed by Graham et al. [4] and the problem is denoted as $F//C_{\max}$.

The FSP is known to be $\mathcal{NP}$-complete in the strong sense when $m \geqslant 3$ (see [5]), if $m=2$ Johnson's algorithm [6], obtains an optimal solution in polynomial time. In general $(n!)^m$ schedules have to be considered ($(n!)^{m-2}$ for $C_{\max}$ criterion).

This paper deals with a simplification of the FSP, which is the permutation FSP or PFSP. In the PFSP *job passing* is not permitted, i.e. the processing sequence of the jobs is the same for all machines. Under this consideration $n!$ schedules are possible and the problem is then denoted as $n/m/P/F_{\max}$ or as $F/prmu/C_{\max}$ (see [2]).

Some authors have claimed that genetic algorithms (GAs) show inferior performance in the PFSP to simulated annealing or tabu search (see for example [7]) and that in order to obtain better results, the genetic algorithm (GA) has to be hybridized with some form of local search. In this paper, we show an advanced GA that does not need hybridization to outperform other well known methods. The proposed algorithm includes some original procedures and operators that have not been applied to the PFSP before. These include four new types of crossover operators that have proved to be superior to many other crossover operators tested, a new generational scheme and a common restarting method. In a second step, the proposed GA is hybridized with a local search based in an insertion neighborhood. Both proposed algorithms are compared against some of the best techniques known so far, including GAs, simulated annealing, tabu search and other recent metaheuristic techniques. For the tests we use the standard benchmark of Taillard [8] that is composed of 120 different problem instances ranging from 20 jobs and 5 machines to 500 jobs and 20 machines. This benchmark contains some instances that have proved to be very difficult to solve in the past 10 years.

The rest of the paper is organized as follows: In Section 2 we present an overview of the existing methods for the PFSP. Section 3 describes in detail the new proposed GA. The proposed hybrid GA is explained in Section 4. In Section 5, the proposed GAs are calibrated by means of the design of experiments (DOE) approach. An extensive comparison of both algorithms with other existing methods is given in Section 6. Finally, in Section 7 we provide some conclusions about the study, along with some future research directions.

2. Existing methods for the PFSP

Generally, the PFSP can be solved with either exact or heuristic methods. Exact methods are only practicable for small instances (less than 15–20 jobs) and even in that case, solution times tend to be high. However, some forms of exact techniques can be used for obtaining optimal solutions for large instances when starting them from high quality, near-optimal solutions obtained by advanced metaheuristics. Researchers have mainly focused their efforts towards heuristic approaches. Heuristics for the PFSP can be divided into constructive and improvement methods. The former are techniques that build a feasible schedule from scratch, and the latter are algorithms that seek to improve a previously generated schedule. There are many constructive heuristics available (for a comprehensive review see [9]). One of the earliest methods is the well known Johnson's algorithm [6]. The heuristics by Palmer [10], Gupta [11], Campbell et al. [12] (CDS) and Dannenbring's Rapid access (RA) procedure [13], are also good examples of constructive methods. It is commonly agreed that the Nawaz et al. heuristic [14] (NEH) is the most effective constructive heuristic for the PFSP. However, there are more recent methods published, for example the heuristics by Koulamas [15] and Davoud Pour [16].

For the improvement heuristics, Dannenbring proposed two different methods, called RACS and RAES. The algorithm by Ho and Chang [17] and the more recent improvement heuristic by Suliman [18] are also good examples.

Metaheuristics can also be considered as improvement heuristics. Within this type of techniques we find GAs, simulated annealing, tabu search and other procedures or hybrid methods.

The first proposed metaheuristics for the FPSP are the simulated annealing algorithms by Osman and Potts [19] and Ogbu and Smith [20]. Widmer and Hertz [21], Taillard [22], Reeves [23] and Nowicki and Smutnicki [24] demonstrated different tabu search approaches. Other algorithms are the path-based method of Werner [25] or the iterated local search of Stützle [26]. Recently, Rajendran and Ziegler [27] have proposed two very effective ant-colony optimization algorithms and Grabowski and Wodecki [28] a very fast tabu search approach.

We focus now on work dealing with GAs and the PFSP. One of the earliest algorithms was proposed by Chen et al. [29]. The initial population in this algorithm is generated by using several heuristic rules. The first $m-1$ population members are generated by the $m-1$ sequences obtained after applying the CDS heuristic of Campbell et al. [12], the mth member is obtained from the *rapid access* (RA) heuristic of Dannenbring. The remaining members are generated from simple job exchanges of the already generated sequences. Only crossover is applied, there is no mutation. The crossover operator used is the *partially mapped crossover* (PMX) by Goldberg and Lingle [30]. Reeves [31] also proposed a GA. This algorithm uses a different

generational scheme, called "termination with prejudice" in which the offspring generated after each mating do not replace the parents but members of the population with a fitness value below average. The algorithm uses a C1 crossover, essentially similar to the *one-point order crossover*. Another remarkable feature of the algorithm is the adaptive mutation used. Reeves' algorithm also initializes the population by using heuristics. In this case, one of the population members is generated by applying the NEH heuristic. Murata et al. [7] proposed a hybrid GA with a *two-point order crossover*, a shift mutation and elitism strategy. The algorithm is hybridized with local search, which resulted in a clear performance gain over a non-hybrid version. Another hybrid GA is that of Reeves and Yamada [32]. In this case a special crossover, called *multi-step crossover fusion* or MSXF is used, coalescing a typical crossover operator with local search. Ponnambalam et al. [33] evaluate a GA with a *generalized position crossover* or GPX crossover, a shift mutation and random population initialization. And more recently, Aldowaisan and Allahvedi [34] have proposed a simple yet effective GA for the permutation flowshop with no-wait constraints.

All the previously cited work on GAs lacks a methodological approach to obtain the correct choice of operators and parameters. Usually, authors make use of short computer simulations or preliminary experiments to set parameters on a "one factor at a time" basis, i.e. changing one parameter while maintaining the remaining factors unaltered. While this approach might be useful when setting some of the algorithms' parameters, it has several shortcomings when setting some of the important operators (like crossover operator or population size). In this paper we use a more comprehensive approach for calibrating the operators and parameters.

3. Proposed GA

In a GA every individual or *chromosome* is encoded into a structure that represents its properties. The set of initial individuals form the *population*. The population is evaluated and all individuals are assigned a *fitness* value, the higher this value, the better the individual. Then the population undergoes a series of operations and the individuals in it *evolve* until some stopping criterion is met. A complete iteration of a GA is called a *generation* and can be briefly described as follows: a *selection* mechanism picks individuals of the current population in such a way that an individual's chance of being selected increases with the fitness value. The selected individuals mate and generate new individuals, called the *offspring*. After the matting process (called *crossover*), some offsprings might suffer a *mutation*. Afterwards, the new population is evaluated again and the whole process is repeated (see, [35,36]).

The effectiveness of GAs greatly depends on the correct choice of the encoding, selection, crossover and mutation operators, as well as the probabilities by which they are applied (see [37]). In this work we will use the DOE approach (see [38]) to set the parameters and operators of the GA. DOE is a structured, organized method for determining the relationship between factors affecting the output of a process. In our case, the factors will all be operators and parameters of the GA and the output will be the relative effectiveness of the GA. The application of the DOE approach to GAs was proposed by Bagchi and Deb [39], and for GAs applied to the PFSP by Jain and Bagchi [40]. However, the experiments shown are rather limited (8 runs maximum) and much more information can be gathered from a more comprehensive experiment. In the following sections we describe all parameters and operators used in the proposed GA.

3.1. Solutions encoding and population initialization

The most frequently used encoding for the PFSP is a simple permutation of the jobs. The relative order of the jobs in the permutation indicates the processing order of the jobs by the machines in the shop. Traditionally, in a GA, the initial population is generated randomly. However, a direct conclusion from the reviewed papers dealing with the PFSP and GAs is that with a random initialization of the population no good results are obtained. The initialization procedure in such a hard combinatorial problem has to be made with great care, to ensure convergence to desirable, better makespans in a reasonable amount of time. The initialization proposed by Chen et al. [29] and Jain and Bagchi [40] creates a population where there are few differences between individuals (Jain and Bagchi show some GAs where 100% of the population is generated from Palmer's heuristic). With such a homogeneous population, premature convergence problems are likely to occur.

The initialization proposed by Reeves [31] (using a NEH generated member) results in a much more effective algorithm. In order to better understand this initialization procedure we are going to briefly describe the NEH procedure, which 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:
$$\forall \text{ job } i,\ i = 1, \ldots, n,\ \ P_i = \sum_{j=1}^{m} p_{ij}.$$
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 already built sequence would contain the first three jobs of the sorted list calculated in step 2, then, the fourth

Table 1
Study of solutions and makespans of Carlier instances (car1–car8)

Instance	n	m	Total different solutions ($n!$)	Total different $C_{\max}$ ($T_{C_{\max}}$)	Ratio $round(\frac{n!}{T_{C_{\max}}})$	# optimum solutions
car7	7	7	5040	1693	3	1
car6	8	9	40,320	2873	14	1
car8	8	8	40,320	1996	20	1
car5	10	6	3,628,800	4119	881	3
car1	11	5	39,916,800	4150	9619	8106
car3	12	5	479,001,600	4667	102,636	18
car2	13	4	6,227,020,800	4562	1,364,976	9690
car4[a]	14	4	87,178,291,200	5030	17,331,668	561,256

[a]More than 15 days of CPU time were needed to solve this instance.

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.

Our initialization procedure is based on this NEH heuristic and on a modification of this heuristic that can be explained as follows: After having ordered the jobs in step 2 we simply pick two random jobs from the ordered list and exchange them for the two first jobs. Then we proceed with the rest of steps 2 and 3. Depending on the choice of the two initial jobs considered for step 2, we will have a different final schedule. In this way, we have an almost limitless supply of very good initial sequences.

We propose the following initialization procedure: the first member of the population is generated by the standard NEH heuristic (just like in Reeves' GA), then up to $B_i\%$ of the initial population is filled with individuals generated from the modified NEH heuristic. The remaining $(100 - B_i)\%$ of the population is filled with completely randomly generated sequences. In this way, we ensure that a $B_i\%$ of the population is formed by fit members.

In preliminary tests, this initialization proved to be superior to the initializations of Reeves and Chen et al. The difference was so clear than no statistical test was needed to ascertain the correct initialization method.

3.2. Selection mechanism and generational scheme

For the selection of parents, we have chosen two classical selection schemes, namely ranking and tournament selection (see [35,36]).

We call a generational scheme the process by which new individuals in a new generation replace old members from the previous generation. Preliminary experiments showed that a *steady state* GA where the offspring replaced the worst individuals in the population was clearly superior to a regular *elitist generational* GA where offspring directly replaced parents.

Let the worst individual in a population be denoted as p_{worst} and its makespan as c_{worst}. We impose several constraints as to when offspring can replace p_{worst}. First, a given offspring can only replace p_{worst} if its makespan is lower than c_{worst}. This initial approach, together with the high selection pressure of the two selection mechanisms considered, resulted in a premature convergence of the population; the main reason being that the fittest members are often selected and the offspring generated replace unfit members. After just a few generations, all the population individuals were essentially similar. To overcome this problem, we devised a new mechanism for determining when an offspring can replace p_{worst}. A new individual will only replace p_{worst} if its makespan is better than c_{worst} and the sequence of the individual is unique, i.e. the sequence is not repeated in the population. Note that there can be many different sequences with the same makespan.

One could think that the same outcome can be obtained by examining the makespan value of the offspring and not allowing repeated makespans in the population (this approach is used in Reeves and Yamada [32]). After all, how many different solutions with equal makespan are there? To answer this we have examined the set of 8 instances of Carlier (car1–car8) taken from the OR-library (`http://mscmga.ms.ic.ac.uk/jeb/orlib/flowshopinfo.html`). For every instance, all the possible solutions ($n!$) are generated and their corresponding makespans calculated and stored. We chose Carlier's benchmark because this benchmark can be solved to optimality, allowing us to show the relevance of our proposed generational scheme. Table 1 shows, from the total possible solutions, the number of different makespans and the number of optimal makespans as well as other data.

We can observe that for larger instances there are many different optimum sequences. Also, there are far fewer different makespan values than possible solutions. For example, instance car1 represents an 11 job, five machine problem and therefore has $11! = 39,916,800$ different permutation schedules. Among that many schedules only 4150 different makespans can be found. This means that all schedules are clustered into 4150 groups with roughly 9619 different permutations with identical $C_{\max}$ value. And, even more relevant, for this instance there are as many as 8106 different

optimum sequences. Disallowing individuals just by judging the makespan and not the entire sequence is not advisable since we would be losing diversity in the genetic process.

3.3. Crossover

The genetic crossover operator generates new sequences or offspring by combining two other sequences or parents. The goal is to generate "better" offspring, i.e. to create better sequences after crossing the parents. Many different general and specific crossover operators have been proposed for the PFSP. There should be neither repeated nor missing elements otherwise the sequence would be illegal. There are several available operators suitable for permutation encodings that are commonly used in published works:

- PMX from Goldberg and Lingle [30].
- OX or *order crossover*, proposed by Davis [41].
- UOB or *uniform order based*, which is a mixture of uniform crossover and order based crossover (see [36]).
- OP or *one-point order crossover*. Comes after mixing the ideas of the one point crossover and the order crossover (see [36]).
- GPX or *generalized position crossover* by [42].
- TP or *two-point order crossover*. Again, it comes from the original two point crossover and the order crossover (see [36]).

Initial simulations showed that crossover often resulted in offspring with worse makespan values than parents. A closer study revealed that this behavior could be explained by the fact that crossover tends to disrupt building blocks, especially in the latter stages of the algorithm. As the proposed generational scheme avoids the occurrence of identical solutions in the population, we can study the diversity in the population and how similar the different individuals are. We found that, after a few generations, there were many similar "blocks" of jobs within the individuals' sequences. After crossover these similar blocks or "building blocks" were broken apart many times in the offspring therefore resulting in worse makespan values. This outcome could be observed in all aforementioned crossover operators.

We propose four new crossover operators for the PFSP that try to overcome this problem. These new operators are based on the idea of identifying and maintaining those building blocks in the crossover. In this way similar blocks or occurrences of jobs in both parents have to be passed over to offspring unaltered. If there are no similar blocks in the parents (as in the initial stage of the algorithm) the crossover operators proposed will behave like the one point order crossover or the two-point order crossover, depending on the case.

The first operator is called "Similar Job Order Crossover" or SJOX and can be explained as follows: First, both parents are examined on a position-by-position basis. Identical jobs at the same positions are copied over to both offspring (Fig. 1(a)). Then, each offspring directly inherits all jobs from one of the parents up to a randomly chosen cut point. That is to say, Offspring 1 inherits directly from Parent 1 and Offspring 2 from Parent 2 (Fig. 1(b)). Lastly, the missing elements of each offspring are copied in the relative order of the other parent (Fig. 1(c)).

In Fig. 1 we can see an example taken out from instance ta001 from Taillard's benchmark. In this case Parents's permutations yield makespan values of 1301 and 1305, respectively, in this instance. These two parents show four identical blocks that are directly copied to the offspring. After crossover, the generated offspring have a $C_{\max}$ of 1293 and 1286, respectively.

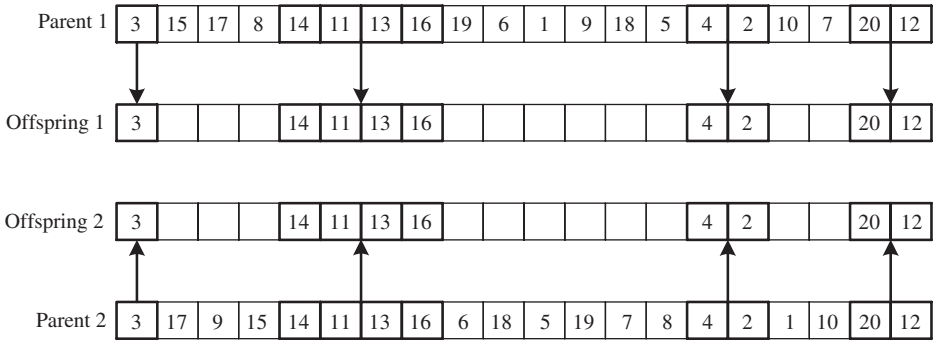
Another alternative comes after the careful examination of the SJOX operator. There might be situations in which there can be several, non-consecutive, identical occurrences in both parents. It can be argued whether these alternated similar positions in parents form a block or not. Thus, we propose another crossover operator, called "Similar Block Order Crossover" or SBOX. In this case, the first step of the SJOX crossover is modified in the following way: We consider blocks of at least two consecutive identical jobs and only those identical blocks that occupy the same positions in both parents are directly copied to offspring. The whole operator is depicted in Fig. 2, showing another example of the same ta001 instance.

Notice how the three similar "isolated" occurrences of jobs 13, 18 and 2 in both parents, located at positions 7, 12 and 14, respectively, are not copied to the offspring (Fig. 2(a)). Notice also how the two different offspring generated do not have jobs 18 and 2 in the same positions as the parents do (Fig. 2(c)).

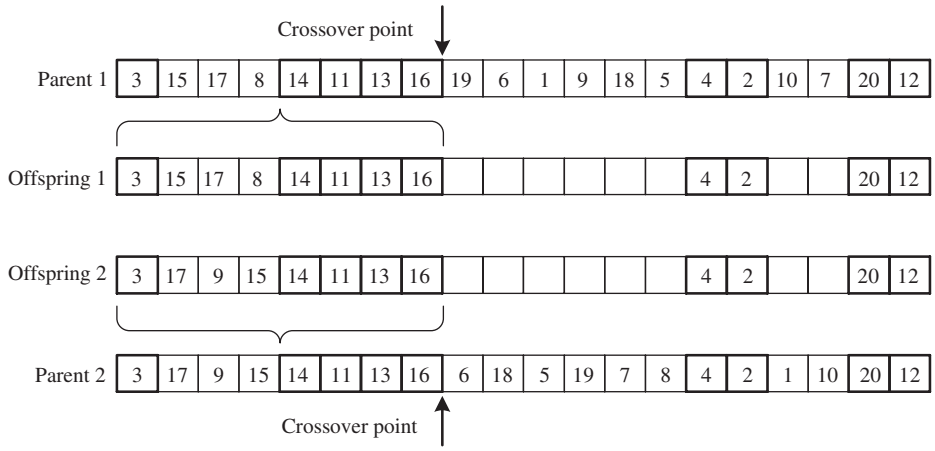
The third and fourth proposed crossover operators are similar to the two previously stated operators. The third operator is called "Similar Job 2-Point Order Crossover" or SJ2OX. The main difference with the SJOX crossover is that in the second step, two random cut points are taken and the section between these two points is directly copied to the children. Therefore, instead of copying up to the cut point from the parents, the whole part between the two cut points is copied. The last proposed operator joins the ideas of the SJ2OX and SBOX crossovers. This crossover is referred to as "Similar Block 2-Point Order Crossover" or SB2OX. In this case, only blocks of at least two consecutive jobs are directly copied to children as in the SBOX crossover in the first step and two cut points are considered in the second step as in the SJ2OX crossover.

3.4. Mutation

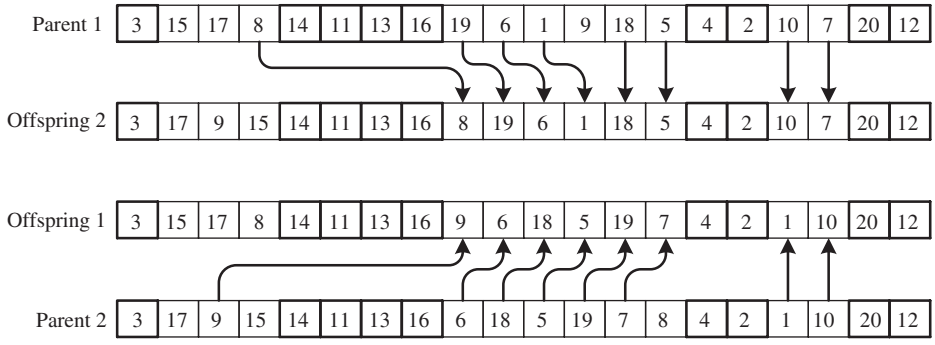
GAs incorporate a mutation operator mainly to avoid convergence to local optimum and to reintroduce lost genetic material and variability in the population. This operator can also be seen as a simple form of local search.



(a) First, the common jobs in both parents are copied over to the offspring.



(b) Then, jobs up to the cut point are inherited from the direct parent.



(c) The missing elements are copied in the relative order of the other parent.

Fig. 1. Similar job order crossover (SJOX).

By mutating an individual we slightly change the sequence, thus allowing a new but similar permutation. Mainly three different mutation operators are proposed in the literature for permutation encodings:

- *SWAP mutation*: Two randomly selected positions are chosen and their corresponding jobs swapped.
- *POSITION mutation*: It is a specific case of the SWAP mutation where two adjacent jobs are swapped.
- *SHIFT mutation*: In this case, a randomly picked position in the sequence is relocated to another randomly picked position. The jobs between these two positions move along.

In this paper, we apply the mutation probability to each position in the sequence. The insertion points and SWAP partners are chosen randomly according to a uniform

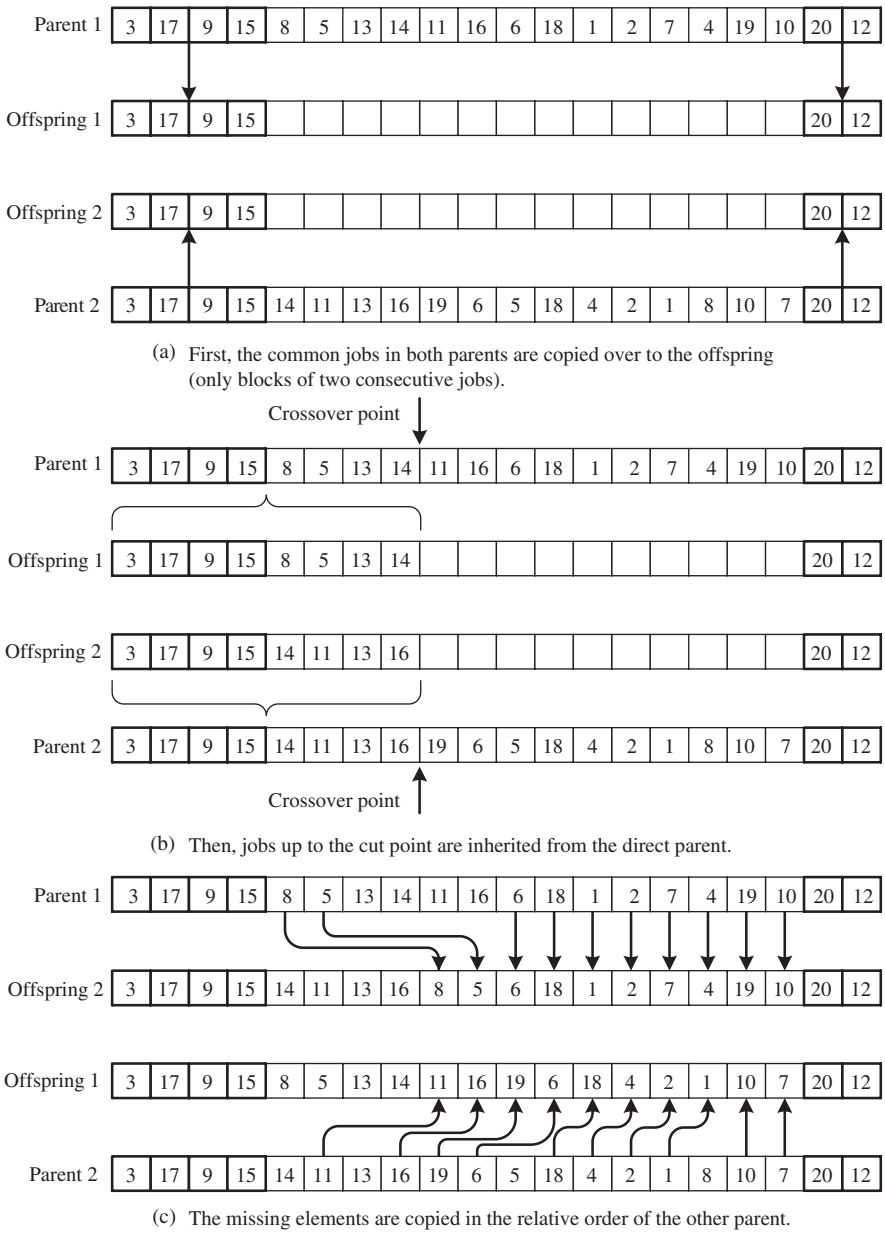


Fig. 2. Similar block order crossover (SBOX) crossover.

distribution among all possible positions. More detailed information about these mutation operators can be found in Michalewicz [36]. In preliminary tests, SHIFT mutation proved to be clearly superior to the other two mutation operators. This, together with the fact that other authors achieved the same results (see [7,31]), motivated the use of SHIFT mutation over the other two methods.

3.5. Restart scheme

In a GA the population evolves over generations. At some given time the population can achieve a sufficiently low diversity for the process to stall around a local optimum. (This fact is not to be confused with the premature convergence problem). To overcome this problem it is common to use

some restart mechanism in GAs. We apply a restart mechanism based on the ideas of a similar scheme used by the authors in a related research [43]. This works as follows:

1. At each generation i, store the minimum makespan, mak_i.
2. If $mak_i = mak_{i-1}$ then make $countmak = countmak + 1$. Otherwise make $countmak = 0$.
3. If $countmak > G_r$ then apply the following procedure:

- Sort the population in ascending order of C_{max}.
- Skip the first 20% individuals from the sorted list (the best individuals).
- From the remaining 80% individuals, 50% of them are replaced by simple SHIFT mutations of the first 20% best individuals (one single mutation). The 25% are replaced by newly generated sequences from the modified NEH procedure and the remaining 25% are replaced by newly randomly generated schedules.
- $countmak = 0$;

With this procedure, every time the lowest makespan in the population does not change for more than G_r generations, the restart procedure will replace all but the 20% best individuals in the population hoping to reintroduce diversity in the population and to escape from local optimum.

4. Hybrid GA

The idea of adding an improvement phase to a GA has been widely exploited before. For example, Murata et al. [7] suggest an improvement phase by applying a local search step before selection and crossover in a GA. The drawback of this approach is that applying local search to all individuals in every generation results in a very slow GA. Our proposal is to apply a local search after selection, crossover and mutation, but not to all individuals in the population. We define an "enhancement probability" or P_{enh} as follows: We draw a random uniformly distributed number between 0 an 1 and if this number is less than or equal to P_{enh} the individual will undergo local search.

For the improvement phase we have several alternatives. We can apply the improvement heuristics of Dannenbring [13], i.e. *rapid access with close order search* (RACS) or *rapid access with extensive search* (RAES). We can also apply Ho and Chang's [17] gap improvement heuristic or some other form of local search.

A simpler and more interesting form of local search follows the ideas of the NEH heuristic. In the last iteration in step 3 of the NEH schedule building process, the remaining job of the ordered job list is inserted in all possible n positions of a sequence that is $n-1$ jobs long and the best schedule among the n candidates is given as a final result. This is no more than the insertion neighborhood, which has been regarded as adequate for the PFSP (see [19,22] or [24]). Given a permutation π of jobs, the insertion neighborhood of π is obtained after considering all the possible pairs of positions $j,\ k \in \{1,\ldots,n\}$ of π, where the job at position j is removed from π and reinserted at position k ($j \neq k$). The resulting neighbor from such a move is $\pi' = (\pi(1),\ldots,\pi(j-1),\pi(j+1),\ldots,\pi(k),\pi(j),\pi(k+1),\ldots,\pi(n))$ when $j < k$, or $\pi' = (\pi(1),\ldots,\pi(k-1),\pi(j),\pi(k+1),\ldots,\pi(j-1),\pi(j+1),\ldots,\pi(n))$ when $j > k$. Every permutation π has $(n-1)^2$ neighbors which can be evaluated in $\mathcal{O}(n^2m)$ (see [22]).

With the insertion neighborhood we use a first improvement local search where all the jobs are extracted from the sequence at random and reinserted in all the possible positions. When a better schedule is found, it is retained and the process continues until all jobs have been examined. If after having inserted all the n jobs we have improved the schedule, the process is repeated again. The local search phase stops when a local optimum is found, i.e. if after having examined all the neighbors, no improvements are found, the local search ends.

With this form of local search, we propose a new GA resulting from the hybridization of the GA detailed in Section 3. After selection, crossover and mutation we apply the local search according to P_{enh}. The new individual obtained after the enhancement is accepted if it has a lower makespan than the non-enhanced individual. After some experimentation we realized that, for low values of P_{enh} the hybridization had a moderate impact on the GAs effectiveness. So, instead of raising P_{enh} (and thus making the algorithm more effective but slower) we thought that, after each generation, the best individual of the population should be enhanced by a probability of $2 \cdot P_{enh}$. It is important to remark that once an individual undergoes local search, it becomes a local optimum with regards to the insertion neighborhood. This means that additional local searches will never improve the individual. Therefore, whenever an individual undergoes local search it is marked as "improved" and will never undergo local search again until disregarded from the population. After having described the proposed GA and the hybrid GA we give a pseudocode description in Fig. 3.

5. Experimental calibration of the algorithms

In this section, we study the behavior of the different operators and parameters of the proposed GAs. All different combinations of the aforementioned factors and parameters yield many alternative GAs. In order to calibrate the algorithms, we have chosen a full factorial design in which all possible combinations of the following factors are tested:

- Selection type: 2 levels (Ranking and Tournament);
- Crossover type: 8 levels (OP, OX, PMX, SB2OX, SBOX, SJ2OX, SJOX and TP);
- Crossover probability (P_c): 5 levels (0,0, 0.1, 0.2, 0.3 and 0.4);

```
popcount:=0; countmak:=0;
population:=Initialize_population(); // Population initialization
Evaluate_individual(population);
MinM:=Min_Makespan(population); // Store minimum makespan of the population
While NOT(TerminationCriterion) do // Main loop
begin
    // Selection
    Parent1:=Selection_Tournament(population);
    Parent2:=Selection_Tournament(population);
    (Child1,Child2):=crossover_SBOX(Parent1,Parent2, P_c) // Crossover
    // Mutation
    Child1:=mutation_SHIFT(Child1, P_m);
    Child2:=mutation_SHIFT(Child2, P_m);
    // Enhancement (only HGA version)
    Child1:=enhance(Child1, P_enh);
    Child2:=enhance(Child2, P_enh);
    Evaluate_individual(Child1,Child2); // New offspring only accepted if better
    // Generational scheme (repeat this procedure for Child2)
     c_worst:=Max_Makespan(population); // Store the worst makespan
     p_worst:=Max(population); // Store the worst individual in the population
    if ( (Makespan(Child1)< c_worst ) AND (
        (Makespan(Child1)<>Makespan(population)) OR (Child unique) ) ) then
            population_ p_worst :=Child1; // replace if better and unique
    popcount:=popcount+2;
    // If we are in a new generation the best individual should be enhanced
    if popcount:= P_size then
        population_best:=Search best individual in the population that is not a local optimum;
        enhance(population_best,2 · P_enh); // (only HGA version)
        if MinM=Min_Makespan(population) then countmak:=countmak+1;
            else countmak:=0; // Controlling the number of generations without improvement
                MinM:=Min_Makespan(population);
        popcount:=0;
    // Restart scheme
    if countmak>G_r then
        countmak:=0;
        population:=Restart(population);
    Evaluate_individual(population);
end;
Return Min_Makespan(population);
```

Fig. 3. Pseudocode for the proposed GAs.

- Mutation probability (P_{m}): 4 levels (0.0, 0.005, 0.01 and 0.015);
- Population size (P_{size}): 4 levels (20, 30, 40 and 50);
- Restart (G_{r}): 3 levels (25, 50 and 75);
- Enhancement Probability (P_{enh}): 4 levels (0.025, 0.05, 0.075 and 0.1).

All the cited factors result in a total of $2 \times 8 \times 5 \times 4 \times 4 \times 3 \times 4 = 15,360$ different combinations and thus, 15,360 different genetic algorithms. Note that if we fix P_{enh} to 0.0 we are in the case of the non-hybrid GA which will be simply referred to as GA_RMA. The version with local search will be referred to as HGA_RMA (i.e. $P_{\mathrm{enh}} = 0.025$, 0.05, 0.075 or 0.01).

Every algorithm is tested with a new set of PFSP instances randomly generated using the procedure described in Taillard [8]. The set of instances comprises 68 combinations of n and m, being $n = \{20, 50, 80, \ldots, 440, 470, 500\}$

and $m = \{5, 10, 15, 20\}$ with the processing times uniformly distributed between 1 and 99. There are two replicates for each combination thus summing up for 136 instances. This set of instances contains more combinations of n and m than Taillard's and does not have orthogonality problems.

The stopping criterion used when testing all instances with the algorithms is set to a CPU time limit fixed to $n(m/2)20$ ms. This stopping criterion allows for more time as the number of jobs or machines increases. All experiments were carried out on a cluster of several PC/AT computers with Athlon XP 1600+ processors (running at 1400 MHz) and 512 MBytes of main memory. The response variable of the experiment is then calculated with the following expression:

$$\text{relative percentage deviation (RPD)} = \frac{\text{Some}_{\text{sol}} - \text{LB}}{\text{LB}} \times 100, \quad (1)$$

where Some_{sol} is the solution obtained by a given algorithm alternative on a given instance out of the 136 and LB is a lower bound for the PFSP calculated as in Taillard [8] for that specific instance. The response variable is, therefore, the average percentage increase over the lower bound for each instance. The resulting experiment was analyzed by means of a multifactor analysis of variance (ANOVA) technique where n and m are also considered as non-controllable factors. Since the ANOVA is parametric, one needs to check the three main hypotheses which are normality, homoskedasticity and independence of the residuals. The residuals resulting from the experimental data were analyzed and all three hypotheses were satisfied. Normality can be studied by drawing a Quantile–Quantile plot of the residuals or by checking how the residuals follow the theoretical normal distribution. We can also use the chi-square, Kolgomorov–Smirnov or Shapiro–Wilks numerical tests for normality. As regards homocedasticity, one can study the dispersion of the residuals when compared to the different levels of all factors. Barlett's numerical test can also be used. Finally, the independence of the residuals is best studied when plotting the residuals against time (see [38]).

The resulting ANOVA has many degrees of freedom. One has to be careful when analyzing results of an experiment with such large sample sizes. The p-values is a component of the ANOVA that serves as a measure of statistical significance. That is, if the p-values is less than or equal to a desired α (or the error probability of rejecting a hypothesis when it is true, also called Type I error) then it is assumed that there is a statistically significant difference between the levels of the factor or interaction considered. The problem is that as the number of experiences grows, the p-values decreases, as more data will lead to the detection of more statistically significant differences, although these differences might be negligible (see [44]). In our case, analyzing the p-values is of little use, since almost all p-values are very close to zero. We focus on the F-ratio, which is the ratio between the variance explained by a factor and the unexplained variance. The greater this ratio, the more effect the factor will have over the response variable. Also, all interactions of more than two factors have been disregarded as their F-ratios were very small.

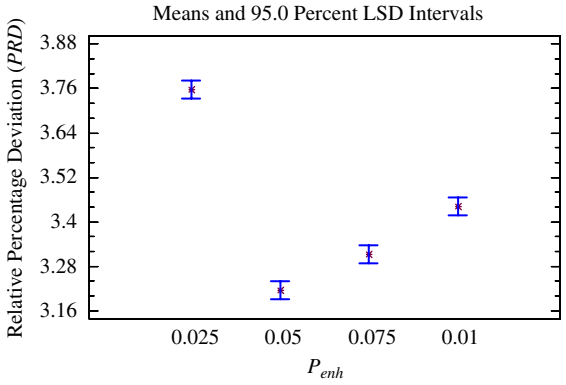


Fig. 4. Means plot for P_{enh} or probability of enhancement factor.

The parameter setting procedure is as follows: We begin by selecting the factor or interaction that has the greatest F-ratio and, by analyzing its different levels in a means plot we can tell which level best suits the response variable and fix the factor at this desired level. Then we pick the second highest F-ratio and apply the same procedure. We finish when all simple factors are fixed to some level.

From the results of the ANOVA analysis, by far, the greatest F-ratio corresponds to both non-controllable factors n and m. This is clearly expected since difficult instances yield greater deviations from the lower bounds for the proposed algorithms (and as is known, the difficulty depends on n and m). Since we cannot control the effect of n and m (these two parameters are input data) we focus on the following factors. The third biggest F-ratio value corresponds to the factor P_{enh} or enhancement probability. We can study the average performance of all the algorithms grouped by the levels of the factor P_{enh}. This is normally done with a means plot along with a multiple comparison test, which in our case is the least significative difference or LSD intervals (at the 95% confidence level). This is all shown in Fig. 4.

It can be observed that a low P_{enh} value of 0.025 yields significantly worse results. A value for P_{enh} of 5% is the best.

The next factor in order of importance is the type of selection. The means plot for this factor can be observed in Fig. 5.

As we can see, there is a clear statistically significant difference between Tournament and Ranking selection schemes and the former results in a better performing GA. If we look closely at the plotted means we find that, on average, the algorithms with ranking selection are around a 3.59% average increase over the lower bound whereas the algorithms

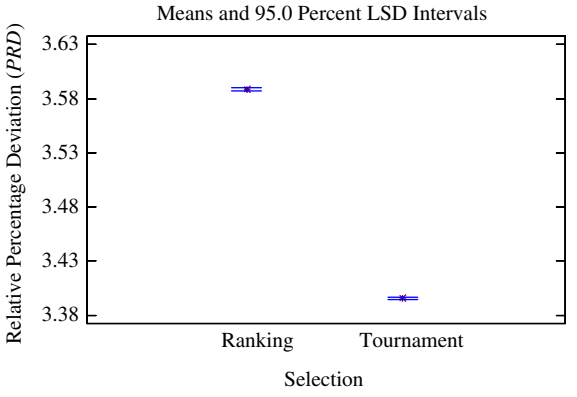


Fig. 5. Means plot for the type of selection factor.

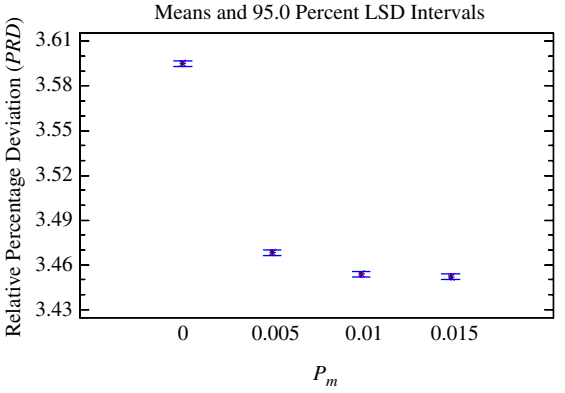


Fig. 6. Means plot for P_{m} or probability of mutation factor.

with tournament selection are about 3.4%. This difference, while being significant from the statistical point of view, has little relevance in reality. A difference of only 0.19% is not large. As we will see from the rest of the statistical analysis, we can advance the fact that our proposed algorithms are robust as regards the choice of parameters and operators.

The next greatest F-ratio corresponds to the probability of mutation or P_{m}. Fig. 6 shows the corresponding means plot.

It can be seen that not using mutation results in a clearly worse algorithm. On the other hand, mutation rates of 1% or 1.5% are statistically similar. Again, as in the selection case, the differences in the response variable between the different mutation probabilities are low, supporting the idea of robustness.

We proceed with the analysis and fix every factor to the most interesting level. We skip the remaining analysis but we stop to examine the behavior of the different crossover operators tested. This is shown in Fig. 7.

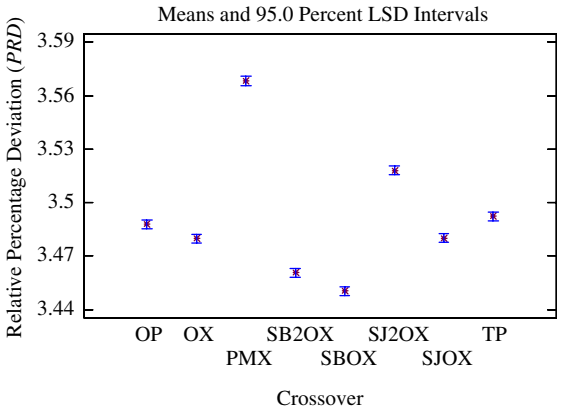


Fig. 7. Means plot for the type of crossover operator factor.

There are statistically significant differences between the eight considered crossover operators. Furthermore, the four proposed crossover operators manage to outperform the classical PMX operator which has been regarded as a fine crossover operator for the PFSP by some authors (see [29,35]). The SB2OX and SBOX operators produce better results than all others, including the OX, OP, and TP. This confirms the idea that transferring similar blocks in parents directly to offspring results in a better performance.

After having completed the calibration of the GA by means of the DOE approach and ANOVA analysis we can summarize the different operators and factors of the final hybrid GA or HGA_RMA (we include other factors that were fixed before the experiment); Selection type: Tournament, crossover type: similar block point order crossover (SBOX), crossover probability ($\mathrm{P_c}$): 0.4, mutation type: SHIFT, mutation probability (P_{m}): 0.01, population size ($\mathrm{P_{size}}$): 20, restart ($\mathrm{G_r}$): 25, enhancement probability ($\mathrm{P_{enh}}$): 0.05 and finally, B_i: 25%.

The previous experiment is also valid for calibrating the GA without local search or GA_RMA. In this case we repeat the experiment fixing P_{enh} to 0.0 (i.e. 3840 algorithms only) and carry out another ANOVA analysis with the same procedure as before. The results are essentially similar to the HGA_RMA with the sole exception of the population size, which in the case of the GA_RMA the best level is 50. All other factors remain unchanged.

It can be argued that applying ANOVA represents an over-calibration and that every algorithm after such a calibration would lead to much improved results. It can be demonstrated that this is not the case for our proposed algorithms. Considering the experiment of the HGA_RMA, the highest RPD (averaged across all 136 instances) among all 15,360 algorithms is 3.85% and the lowest 3.22%. This means that the worst possible combination (with really awry parameters like $P_{\mathrm{c}} = P_{\mathrm{m}} = 0$, PMX crossover and so on) is about 19% worse than the best possible combination. The differences among algorithms with more "normal" parameters are even lower.

6. Computational experience

In this section, we are going to compare both proposed GAs, GA_RMA and HGA_RMA with other GAs, simulated annealing, tabu search and some other state-of-the-art techniques for the PFSP.

The compared methods are: the NEH heuristic by Nawaz et al. with the enhancements of Taillard [22] (NEHT), the GA of Chen et al. (GAChen), the simulated annealing of Osman and Potts (SAOP), the tabu search of Widmer and Hertz (Spirit), the GA of Reeves (GAReev), the hybrid GA of Murata et al. (GAMIT), the iterated local search procedure of Stützle (ILS), the GA of Aldowaisan and Allahvedi adapted to the PFSP (GA_AA) and lastly the two recent ant colony algorithms of Rajendran Ziegler [27], referred to as M-MMAS and PACO, respectively.

For the evaluation we are going to use the well known standard benchmark set of Taillard [8] that is composed of 120 different problem instances ranging from 20 jobs and 5 machines to 500 jobs and 20 machines. The benchmark contains 10 repetitions for each considered combination of n and m. We average the results for all the 10 instances in a given combination.

We carry out $R = 5$ independent runs for all algorithms (with the sole exception of NEHT) and the results averaged. The stopping criterion is fixed to a given maximum elapsed CPU time that follows the expression $n(m/2)t$ ms, where $t = 30, 60$ and 90. The choice of this stopping criterion is motivated by the fact that all the algorithms have been coded in the same programming language (Delphi 7.0), use many common functions and structures, and are tested on the same computer; an Intel Pentium IV processor running at 2.8 GHz with 512 MBytes of main memory. The operating system used is Windows XP Professional and the optimization flag present in Delphi 7.0 was enabled. In this situation, all the algorithms are fully comparable.

We have also included, for reasons of comparison, a simple method that generates and evaluates random schedules and returns the best one as a result. This method, which we call RAND, will serve as a "worst case" since we expect a low effectiveness from this algorithm.

For evaluating the different methods (13 in total) we use a similar performance measure to the one given in expression (1):

$$\text{average relative percentage deviation } (\overline{\text{RPD}}) = \sum_{i=1}^{R} \left(\frac{\text{Heu}_{\text{sol}_i} - \text{Best}_{\text{sol}}}{\text{Best}_{\text{sol}}} \times 100 \right) / R, \quad (2)$$

where $\text{Heu}_{\text{sol}_i}$ is the solution given by any of the R repetitions of the considered algorithms and in this case Best_{sol} is either the optimum solution or the lowest known upper bound for Taillard's instances as of late April 2004 (these values are available at Taillard [45]). The results for $t = 30$, 60 and 90 are shown in Table 2.

From the tables we see that the deterministic NEHT heuristic yields a $\overline{\text{RPD}}$ of 3.35%, which is much better than the RAND rule or the metaheuristics Spirit and GAChen. This result holds even in the case of $t = 90$ where all the metaheuristics tested need 450 s to solve the 500×20 instances and the NEHT only needs 84.24 ms on average. It is clear that both metaheuristics would clearly benefit from a NEHT initialization. Other algorithms like SAOP, GAMIT and GA_AA do obtain better results than NEHT and GA_AA seems to be better. The case of SAOP is remarkable, since it obtains very good results even though it starts from a random solution and not a NEHT generated one. The GA of Reeves obtains better solutions resulting in a $\overline{\text{RPD}}$ of less than 1.5% in the case of $t = 90$. However, our proposed GA_RMA provides substantially better results in almost all cases, these differences are up to 43% in the case of $t = 90$. As a matter of fact, the proposed GA_RMA provides better results than all the other four considered genetic algorithms, named GAReev, GAMIT, GAChen and GA_AA.

There are three algorithms that manage less than a 1% $\overline{\text{RPD}}$. These are our proposed HGA_RMA and the two ant colony algorithms of Rajendran and Ziegler. The HGA_RMA is much better than the corresponding version without local search under the same elapsed time as a stopping criterion. Thus the proposed hybridization with local search is both very simple and effective. More precisely, the local search version is more than two times as effective as the version without local search. The ant algorithm PACO is slightly better than the M-MMAS, confirming the previous findings from the original authors. As a conclusion, the best algorithm from the comparison is our proposed HGA_RMA. The difference with the second best algorithm (PACO ant colony) is up to 48.89%.

It is also interesting to check whether these observed differences in the $\overline{\text{RPD}}$ values are indeed statistically significant. We have carried out a single factor design of experiments where we consider a factor "algorithm" and introduce the results given by the best algorithms: ILS, GA_RMA, HGA_RMA, M-MMAS and PACO. The response variable is 120 RPD values for each algorithm. The means plot for the single factor is depicted in Fig. 8 for the case of $t = 90$.

We see that there are no statistically significant differences between our proposed algorithm GA_RMA and ILS. The two ant colony algorithms are also equivalent. From the results we see that our proposed HGA_RMA produces statistically better results than all others.

In the previous evaluation we have considered many of the best known metaheuristics, including several GAs as well as other recent methods. However, the comparison does not consider some of the state-of-the-art metaheuristics like for example the tabu search algorithm of Nowicki and Smutnicki [24] (referred to as TSAB), the recent tabu search of Grabowski and Wodecki [28] (TSGW) or the well-known GA with path relinking of Reeves and Yamada [32] (RY). TSAB has been regarded as one of the most effective algorithms for the PFSP (see [26,46]) and recently Grabowski

Table 2
Average relative percentage deviation ($\overline{\text{RPD}}$) over the optimum solution or lowest known upper bound for Taillard's instances obtained by the methods evaluated

Instance	RAND	NEHT	GA_RMA	ILS	SAOP	Spirit	GAChen	GAReev	GAMIT	**HGA_RMA**	GA_AA	M-MMAS	PACO
$t = 30$													
20 × 5	4.04	3.35	0.24	0.50	1.17	3.91	3.65	0.54	0.84	**0.05**	0.94	0.11	0.20
20 × 10	7.65	5.02	0.62	0.63	2.69	5.41	5.00	1.78	1.96	**0.10**	1.70	0.15	0.32
20 × 20	6.25	3.73	0.37	0.38	2.21	4.51	3.90	1.39	1.66	0.10	1.31	**0.09**	0.21
50 × 5	3.57	0.84	0.06	0.21	0.45	1.99	1.89	0.17	0.30	**0.00**	0.37	0.02	0.08
50 × 10	12.18	5.12	1.79	1.49	3.71	5.95	6.37	2.23	3.50	**0.77**	3.60	1.30	0.90
50 × 20	14.59	6.26	2.67	2.23	4.57	7.64	7.88	3.74	5.07	**1.19**	4.66	2.10	1.46
100 × 5	2.92	0.46	0.07	0.18	0.33	0.98	1.34	0.14	0.25	**0.02**	0.26	0.03	0.04
100 × 10	9.18	2.13	0.65	0.69	1.52	3.13	3.90	0.82	1.54	**0.26**	1.65	0.46	0.35
100 × 20	15.13	5.23	2.78	2.58	4.79	6.65	8.06	3.36	4.99	**1.59**	4.92	2.59	2.17
200 × 10	7.45	1.43	0.43	0.59	1.08	2.08	2.80	0.59	1.14	**0.16**	1.08	0.37	0.26
200 × 20	14.53	4.41	2.35	2.25	4.11	5.00	6.94	2.71	4.19	**1.42**	3.95	2.34	2.00
500 × 20	10.96	2.24	1.43	1.26	2.34	9.87	4.79	1.47	2.68	**0.87**	2.06	1.06	0.98
Average	9.04	3.35	1.12	1.08	2.42	4.76	4.71	1.58	2.34	**0.55**	2.21	0.88	0.75
$t = 60$													
Instance	RAND	NEHT	GA_RMA	ILS	SAOP	Spirit	GAChen	GAReev	GAMIT	**HGA_RMA**	GA_AA	M-MMAS	PACO
20 × 5	3.80	3.35	0.23	0.44	1.30	4.52	4.02	0.51	0.74	**0.03**	0.80	0.08	0.16
20 × 10	7.09	5.02	0.60	0.56	2.50	5.46	5.14	1.67	1.72	**0.09**	1.41	0.09	0.30
20 × 20	5.86	3.73	0.34	0.33	2.29	5.18	3.93	1.41	1.66	**0.07**	1.37	0.07	0.15
50 × 5	3.20	0.84	0.06	0.19	0.46	2.04	2.02	0.20	0.26	**0.01**	0.37	0.02	0.03
50 × 10	11.81	5.12	1.86	1.47	3.41	6.01	6.83	2.26	3.20	**0.64**	3.35	1.14	0.87
50 × 20	14.50	6.26	2.62	2.17	4.46	7.44	7.98	3.71	4.88	**1.07**	4.52	2.06	1.39
100 × 5	2.67	0.46	0.08	0.17	0.30	0.97	1.44	0.12	0.25	**0.01**	0.24	0.02	0.03
100 × 10	8.96	2.13	0.62	0.66	1.45	3.11	3.78	0.74	1.46	**0.23**	1.61	0.42	0.32
100 × 20	14.97	5.23	2.68	2.51	4.41	6.56	8.18	3.25	4.77	**1.33**	4.73	2.50	1.99
200 × 10	7.35	1.43	0.41	0.55	0.99	1.83	2.75	0.50	1.04	**0.13**	1.10	0.32	0.26
200 × 20	14.38	4.41	2.22	2.21	3.90	5.06	7.24	2.65	4.14	**1.30**	4.02	2.18	1.86
500 × 20	10.85	2.24	1.40	1.23	2.24	7.41	4.79	1.38	2.48	**0.76**	1.98	1.09	0.92
Average	8.79	3.35	1.09	1.04	2.31	4.63	4.84	1.53	2.22	**0.47**	2.13	0.83	0.69
$t = 90$													
Instance	RAND	NEHT	GA_RMA	ILS	SAOP	Spirit	GAChen	GAReev	GAMIT	**HGA_RMA**	GA_AA	M-MMAS	PACO
20 × 5	3.79	3.35	0.25	0.33	1.05	4.77	3.51	0.62	0.53	**0.04**	0.84	0.04	0.18
20 × 10	7.08	5.02	0.64	0.52	2.60	5.61	4.99	1.71	1.61	**0.02**	1.42	0.07	0.24
20 × 20	5.72	3.73	0.40	0.28	2.06	4.72	4.24	1.31	1.36	**0.05**	1.23	0.06	0.18
50 × 5	3.12	0.84	0.06	0.18	0.34	2.24	2.34	0.16	0.23	**0.00**	0.34	0.02	0.05
50 × 10	11.60	5.12	1.46	1.45	3.50	5.69	6.92	2.00	3.27	**0.72**	3.30	1.08	0.81

Table 2 (*continued*)

Instance	RAND	NEHT	GA_RMA	ILS	SAOP	Spirit	GAChen	GAReev	GAMIT	**HGA_RMA**	GA_AA	M-MMAS	PACO
50 × 20	14.31	6.26	2.47	2.05	4.66	7.59	7.77	3.58	4.75	**0.99**	4.69	1.93	1.41
100 × 5	2.66	0.46	0.06	0.16	0.30	0.93	1.36	0.11	0.22	**0.01**	0.22	0.02	0.02
100 × 10	8.85	2.13	0.52	0.64	1.34	3.22	3.87	0.67	1.34	**0.16**	1.55	0.39	0.29
100 × 20	14.90	5.23	2.54	2.42	4.49	6.75	8.11	3.12	4.68	**1.30**	4.64	2.42	1.93
200 × 10	7.28	1.43	0.41	0.50	0.94	2.06	2.81	0.41	0.98	**0.14**	0.99	0.30	0.23
200 × 20	14.14	4.41	2.11	2.07	3.67	5.02	7.37	2.54	3.95	**1.26**	3.86	2.15	1.82
500 × 20	10.76	2.24	1.36	1.20	2.20	5.59	4.62	1.33	2.36	**0.69**	2.08	1.02	0.85
Average	8.68	3.35	1.02	0.98	2.26	4.52	4.83	1.46	2.11	**0.45**	2.10	0.79	0.67

Stopping criterion set at $n(n/2)t$ ms CPU time with $t = 30$, 60 and 90.

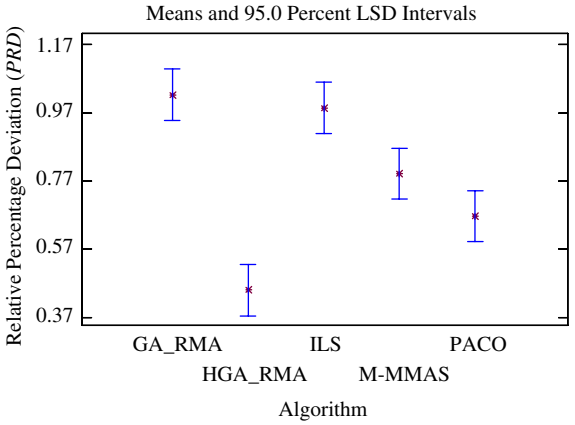


Fig. 8. Means plot for the relative percentage deviation (RPD) for Taillard's benchmark and the best algorithms tested. Stopping criterion set at $n(n/2)t$ ms CPU time with $t = 90$.

and Wodecki have proved their TSGW algorithm to be even faster and more effective. All these algorithms have in common that they are very complex and intricate. They all work with the critical path concept present in the PFSP with the $C_{\max}$ criterion.

After a hard and tedious re-implementation work, our versions of the TSAB and RY algorithms could not match the published results, especially in terms of efficiency.[1] We requested an executable and/or source code to the corresponding authors of the three papers for comparisons. Unfortunately, neither the source code nor the executables were provided. The only way of comparing is therefore against the published results in these papers. Some of these results were obtained by outdated computers with no specified running time, as are the best results from TSAB, or with modern mainframes like in TSGW. Additionally, our proposed HGA_RMA algorithm does not make use of any of the speed-ups and tweaks used in TSAB, TSGW or RY. Implementing these speed-ups would have led to a much faster, albeit much more complex algorithm. Consequently, such a comparison has not been carried out. However, at a computation time penalty, the proposed HGA_RMA algorithm reaches and even surpasses the performance of the state-of-the-art algorithms that include several sophisticated speed-up tricks. Therefore, the advantage of the HGA_RMA is that it is much simpler and does not make use of the critical path concept that is specific to the $C_{\max}$ criterion.

7. Conclusions and future research

In this work we have proposed two new robust genetic algorithms (GAs) for the permutation flowshop scheduling problem (PFSP) under the makespan minimization criterion.

[1] The coding of the TSGW algorithm has not been attempted.

The algorithms include a carefully studied initialization of the population. We have also devised a new generational scheme that enforces strong pressure but at the same time avoids premature convergence of the population. We also present four new crossover operators that have shown better results than the *partially mapped crossover* or PMX as well as many others. Another feature is the use of a common restart scheme that restarts a given portion of the population if the best makespan has not been improved for a given number of generations.

The proposed GA has been tuned by means of a design of experiments (DOE) approach that involves the evaluation of many different algorithm alternatives. For every algorithm we have solved a new set of permutation flowshop instances. The proposed hybrid GA uses a simple form of local search based on the NEH algorithm by Nawaz et al. [14].

We have conducted an extensive comparison of the two proposed GAs against 11 other algorithms including a simple random rule, the well known NEH heuristic, a simulated annealing, a tabu search, four different GAs, an iterated local search procedure and two recent ant colony algorithms. The results obtained show that the hybrid version of the proposed GA outperforms all the other compared algorithms, producing results that are almost 49% better than the second best algorithm considered. The evaluations situate our algorithms as some of the finest and most efficient methods proposed to date for the problem considered while being at the same time simple and easy to implement. Furthermore, the GAs studied could be modified to take into account more realistic aspects of the problem such as sequence-dependent setup times (SDST flowshop), unrelated parallel machines at each stage (general hybrid flowshop) or the existence of due dates.

All the code used in this paper, as well as the proposed algorithms is available upon request.

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).

References

[1] Baker KR. Introduction to sequencing and scheduling. New York: Wiley; 1974.

[2] Pinedo M. Scheduling: theory, algorithms and systems. 2nd ed., Englewood Cliffs, NJ: Prentice Hall; 2002.

[3] Conway RW, Maxwell WL, Miller LW. Theory of scheduling. Reading, MA: Addison-Wesley; 1967.

[4] Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 1979;5:287–326.

[5] Garey MR, Johnson DS, Sethi R. The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1976;1(2):117–29.

[6] Johnson SM. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1954;1:61–8.

[7] Murata T, Ishibuchi H, Tanaka H. Genetic algorithms for flowshop scheduling problems. Computers & Industrial Engineering 1996;30(4):1061–71.

[8] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64:278–85.

[9] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics, European Journal of Operational Research 2005;165:479–94.

[10] Palmer DS. Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum. Operational Research Quarterly 1965;16(1):101–7.

[11] Gupta JND. A functional heuristic algorithm for the flowshop scheduling problem. Operational Research Quarterly 1971;22(1):39–47.

[12] Campbell HG, Dudek RA, Smith ML. A heuristic algorithm for the n job, m machine sequencing problem. Management Science 1970;16(10):B630–7.

[13] Dannenbring DG. An evaluation of flow shop sequencing heuristics. Management Science 1977;23(11):1174–82.

[14] Nawaz M, Enscore EEJ, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA, The International Journal of Management Science 1983;11(1):91–5.

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

[16] Davoud Pour H. A new heuristic for the n-job, m-machine flow-shop problem. Production Planning and Control 2001;12(7):648–53.

[17] Ho JC, Chang Y-L. A new heuristic for the n-job, M-machine flow-shop problem. European Journal of Operational Research 1991;52:194–202.

[18] Suliman SMA. A two-phase heuristic approach to the permutation flow-shop scheduling problem. International Journal of Production Economics 2000;64:143–52.

[19] Osman IH, Potts CN. Simulated annealing for permutation flow-shop scheduling. OMEGA, The International Journal of Management Science 1989;17(6):551–7.

[20] Ogbu FA, Smith DK. The application of the simulated annealing algorithms to the solution of the $n/m/C_{\max}$ flowshop problem. Computers & Operations Research 1990;17(3):243–53.

[21] Widmer M, Hertz A. A new heuristic method for the flow shop sequencing problem. European Journal of Operational Research 1989;41:186–93.

[22] Taillard E. Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research 1990;47:67–74.

[23] Reeves CR. Improving the efficiency of tabu search for machine scheduling problems. Journal of the Operational Research Society 1993;44(4):375–82.

[24] Nowicki E, Smutnicki C. A fast tabu search algorithm for the permutation flow-shop problem. European Journal of Operational Research 1996;91:160–75.

[25] Werner F. On the heuristic solution of the permutation flow shop problem by path algorithms. Computers & Operations Research 1993;20(7):707–22.
[26] Stützle T. Applying iterated local search to the permutation flow shop problem. Technical report, AIDA-98-04, TU Darmstadt, FG Intellektik, 1998.
[27] Rajendran C, Ziegler H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research 2004;155:426–38.
[28] Grabowski J, Wodecki M. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Computers & Operations Research 2004;31:1891–909.
[29] Chen C-L, Vempati VS, Aljaber N. An application of genetic algorithms for flow shop problems. European Journal of Operational Research 1995;80:389–96.
[30] Goldberg D, Lingle Jr R. Alleles, loci, and the traveling salesman problem. In: Grefenstette JJ (Ed.), Proceedings of the first international conference on genetic algorithms and their applications, Hillsdale, NJ, 1985. Lawrence Erlbaum associates, pp. 154–9.
[31] Reeves CR. A genetic algorithm for flowshop sequencing. Computers & Operations Research 1995;22(1):5–13.
[32] Reeves C, Yamada T. Genetic algorithms, path relinking, and the flowshop sequencing problem. Evolutionary Computation 1998;6(1):45–60.
[33] Ponnambalam SG, Aravindan P, Chandrasekaran S. Constructive and improvement flow shop scheduling heuristics: an extensive evaluation. Production Planning and Control 2001;12(4):335–44.
[34] Aldowaisan T, Allahvedi A. New heuristics for no-wait flowshops to minimize makespan. Computers & Operations Research 2003;30:1219–31.
[35] Goldberg DE. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley; 1989.
[36] Michalewicz Z. Genetic algorithms + data structures = evolution programs. 3rd ed., Berlin, Heidelberg: Springer; 1996.
[37] Alcaraz J, Maroto C. A robust genetic algorithm for resource allocation in project scheduling. Annals of Operations Research 2001;102:83–109.
[38] Montgomery DC. Design and analysis of experiments. 5th ed., New York: Wiley; 2000.
[39] Bagchi TP, Deb K. Calibration of GA parameters: The design of experiments approach. Computer Science and Informatics 1996;26(3):46–56.
[40] Jain N, Bagchi TP. Flowshop scheduling by hybridized GA: Some new results. International Journal of Industrial Engineering 2000;7(3):213–23.
[41] L. Davis, Applying adaptative algorithms to epistatic domains, in: Proceedings of the Interanational joint conference on artificial intelligence, pp. 162–4, 1985.
[42] Mattfeld DC. Evolutionary search and the job shop investigations on genetic algorithms for production scheduling. Heidelberg: Physica-Verlag; 1996.
[43] Alcaraz J, Maroto C, Ruiz R. Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. Journal of the Operational Research Society 2003;54:614–26.
[44] Montgomery DC, Runger GC. Applied statistics and probability for engineers. New York: Wiley; 1994.
[45] Taillard E. Summary of best known lower and upper bounds of Taillard's instances. http://ina.eivd.ch/collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html, 2004.
[46] Anderson EJ, Glass CA, Potts CN. Machine scheduling. in: Aarts E, Lenstra JK (Eds.), Local search in combinatorial optimization. Chichester: Wiley; 1997 [Chapter 11] pp. 361–414.