Metaheuristics for Solving a Hybrid Flexible Flowshop Problem with Sequence-Dependent Setup Times

Aymen Sioud(✉), Caroline Gagné, and Marc Gravel

Département d'informatique et de mathématique,
Université du Québec à Chicoutimi, 555 Boulevard Université,
Chicoutimi, Québec G7H 2B1, Canada
{asioud,c3gagne,mgravel}@uqac.ca

Abstract. In this paper, we propose three new metaheuristic implementations to address the problem of minimizing the makespan in a hybrid flexible flowshop with sequence-dependent setup times. The first metaheuristic is a genetic algorithm (GA) embedding two new crossover operators, and the second is an ant colony optimization (ACO) algorithm which incorporates a transition rule featuring lookahead information and past information based on archive concepts such as the multiobjective evolutionary computation. The third metaheuristic is a hybridization (HGA) of the GA and the ACO algorithms. Numerical experiments were performed to compare the performance of the proposed algorithms on different benchmarks from the literature. The algorithms are compared with the best algorithms from the literature. The results indicate that our algorithms generate better solutions than those of the known reference sets.

Keywords: ACO · Genetic Algorithm · Hybrid Metaheuristics · Scheduling · Hybrid flowshop · Makespan · Sequence-dependent setup times

1 Introduction

Among production scheduling systems, the flowshop is one of the most well-studied environments. In this configuration, all jobs follow the same routing scheme. The associated problem can be considered as a basic model for several variants of real problems. Moreover, real production systems rarely employ a single machine at each stage. Therefore, the regular flowshop problem is often extended to a set of usually identical parallel machines at each stage. That is, instead of having a series of machines, we have a series of stages. The goal here is to increase the capacity and the outflow of the production system and to reduce the impact of bottleneck stages on overall shop efficiency. It is also frequent in practice to have optional treatments for products, like polishing or additional decorations in ceramic manufacturing as examples [2,21]. In this latter case some jobs will skip some stages. We obtain thereby the hybrid flexible flowshop.

P. Siarry et al. (Eds.): ICSIBO 2014, LNCS 8472, pp. 9–25, 2014.
DOI: 10.1007/978-3-319-12970-9_2

Furthermore, in many industries such as pharmaceutics, metallurgy, ceramics and automotive manufacturing, we often have setup times on equipment between two different jobs. Many authors assume that setup times are negligible, or a part of the job processing time. But explicit setup times must be included in scheduling decisions in order to model a more realistic variant of hybrid flowshop scheduling problems. These setup times may or may not be sequence-dependent. [6] reported that 70% of industrial activities include dependent setup times. More recently, [3] pointed out in 250 industrial projects that 50% of these projects contain setup dependent times and when these setup times are applied, 92% of the order deadline could be met. Production of good schedules often relies on good management of these setup times [1,29].

The present paper considers the hybrid flexible flowshop problem with sequence-dependent setup times (SDST/HFFS) minimizing the makespan. In accordance with the notation for hybrid flowshops introduced by [26] who extended the well-known three fields notation $\alpha/\beta/\gamma$ of [8], this problem is noted as $((PM)^{(i)})_{i=1}^{m}/F_j, s_{ijk}/C_{max}$. [9] showed that the flowshop with multiple processors (FSMP) problem with only two stages ($m = 2$), which can be considered as a special case of the SDST/HFFS problem, is NP-hard, and therefore the SDST/HFFS problem studied in this paper is also NP-hard.

The $((PM)^{(i)})_{i=1}^{m}/F_j, s_{ijk}/C_{max}$ problem may be defined as a set of N jobs, N=$\{1,\ldots, n\}$, available for processing at time zero on a set of M stages, M=$\{1,\ldots, m\}$. At every stage i, $i \in M$, we have a set of M_i, M_i=$\{1,\ldots, m_i\}$, identical parallel machines. Every machine at each stage can process all the jobs. Each job has to be processed in exactly one of the M_i identical parallel machines at stage i. However, some jobs will skip some stages. F_j denotes the set of stages that the job j, $j \in N$ has to visit. Furthermore, only stage skipping is allowed, so it is not possible for a given job to visit stages {1, 2, 3} and another one to visit stages {3, 2, 1}. p_{ij} denotes the processing time of job j at stage i. Finally, s_{ijk} denotes the setup time between jobs j and k, $k \in N$ at stage i. The optimization criterion is the minimization of the maximum completion time or makespan, which is calculated as $C_{max} = max_{j \in N}\{C_j\}$.

Considering the completion time criterion for a regular flowshop problem, a simple permutation of the jobs in an array constitutes the most widely used encoding for the sequences. Nevertheless, when we handle a hybrid flowshop scheduling problem with this kind of representation, two main decisions have to be taken : (i) determine the job sequence at the beginning of each stage, and (ii) assign jobs to machines at each stage. For the HFFS problem, the job sequence at the first stage is normally determined by the outcome of the scheduling algorithms. For subsequent stages, the jobs are sorted in increasing value of their completion times in the previous stage. Furthermore, in the case of HFFS without setup times, assigning jobs at each stage to the first available machine (FAM), which results in the earliest completion time for the jobs in that stage, represents a possibility as the assigning decision. However, in the case of the SDST/HFFS we use the earliest completion time (ECT) rule which incorporates the incurred setup times between two jobs when calculating the job completion

times. These completion times are calculated as $C_{ij} = max\{C_{i,j-1}, C_{i-1,j}\} + S_{i,j-1,j} + p_{ij}$, where $C_{i,j-1}$ is the completion time of the previous job in the sequence that was assigned to the same machine as job j at stage i. Similarly, $C_{i-1,j}$ is the completion time of job j at the previous stage that this job visited. [17] enhance the ECT rule with the fast earliest completion time rule (FECT). This technique arranges the jobs in the same relative order when they have the same ready times at each stage. The authors show that this technique be very effective in the presence of stage skipping. We will use the FECT rule in the rest of the paper.

In this work, to solve the SDST/HFFS problem, we introduce new crossover operators in a genetic algorithm (GA), a new transition rule in an ant colony optimization algorithm (ACO) and a new hybrid metaheuristic involving the GA and the ACO. These represent the main contribution of this paper. The innovative GA and ACO are designed and developed to adapt to the treated problem. Both are then combined to create a new hybrid metaheuristic GA/ACO with features not seen in the traditional GA and ACO. All the proposed approach are essentially based on adapting their different mechanisms to the specifics of the problem studied.

The body of this paper is organized into five sections. Section 2 provides a brief literature review of the SDST/HFFS problem. Section 3 describes the proposed GA, while Section 4 and Section 5 describe the ACO algorithm and the hybrid genetic algorithm (HGA), respectively. The computational testing and discussion are presented in Section 6. Finally, we conclude with some remarks and future research directions.

2 Literature Review of SDST/HFFS

There is not much published research on the SDST/HFFS problem. To our knowledge, there are only papers proposing heuristics and/or metaheuristics for this problem. [13] introduced dispatching rules based on greedy methods, insertion heuristic and an adaption of Johnson's rule. Later, they [14] formulated an integer programming (IP) model and developed random keys genetic algorithm (RKGA). The results showed that the IP model does not easily solve the SDST/HFFS problem and that the RKGA outperforms the dispatching rules of [13] and other heuristics. All the algorithms are tested on generated problem data. [28] proposed an immune algorithm (IA) which outperform the RKGA of [14]. The authors used a real representation for individuals and the order crossover (OX) as the crossover operator. [16] proposed a simulated annealing (SA) using pair-wise and inverse interchange as moving operators. They also used the Shortest Processing Time Cyclic Heuristic of [14], showing that the SA outperforms the RKGA of [14] and the IA of [28]. [17] proposed a dynamic dispatching rule heuristic and an iterated local search (ILS). They also proposed 960 test instances and compared their approaches to the dispatching rules of [13], the RKGA of [14], the IA of [28] and the GA of [21] which is used for a different problem. The results showed that their ILS with different encoding scheme gives

better results than all the other algorithms. [7] proposed an agent-based genetic algorithm using the Similar Block 2-Point Order Crossover (S2BOX) of [21] and introducing the agent-solution scheme to solve the SDST/HFFS problem.

There are studies about related problems with a more complex setting and/or variations of the SDST/HFFS problem. [21] discussed also the SDST/HFFS problem but they assumed that some machines are not eligible to perform some jobs. They proposed a genetic algorithm using new and classic crossover operators from the literature. [22] proposed two iterated greedy heuristic (IGH) for the SDST/HFFS problem with the objectives of minimizing the makespan and the weighted tardiness. In this paper the authors consider release dates for machines, machine eligibility, possibility of the setup times to be both anticipatory and non-anticipatory, precedence constraints and time lags. [11] proposed three heuristics, based on Shortest Processing Time (SPT), Longest Processing Time (LPT) and the Johnson rule, and two metaheuristics based on a genetic algorithm and simulated annealing, to solve the SDST/HFFS problem with machine availability constraints. [12] proposed an immune algorithm (IA) for solving the SDST/HFFS problem with time lags on the machines comparing it with the IP model on small instances. [18] studied a hybrid flowshop with setup times where no flexibility is considered. They proposed a variation of simulated annealing using the Taguchi method and minimizing the makespan and the maximum tardiness.

3 A Genetic Algorithm for the SDST/HFFS

Genetic algorithms are methods based upon biological mechanisms such as the genetic inheritance laws of Mendel and the natural selection concept of Darwin, where the best adapted species survive. The basic concepts of GAs have been described by [10]. He explains how to add intelligence to a program by using the crossover exchange of genetic material and transfer which is a source of genetic diversity. Indeed, this kind of metaheuristic works with a set of individuals called the population. Every chromosome is evaluated and assigned a fitness value. This evolving process exchanges genetic material and uses crossover and mutation operators to transfer it, generating new individuals called offspring. Selection and replacement processes are applied to reach better individuals over the generations,converging to an optimum in the solution search space. The effectiveness of a GA depends on the choice of its operators and parameters, but also on the specific adaptation to the problem treated. In the following sections, we explain the different choices of the GA's parameters and we describe the proposed crossover operators.

3.1 Population Encoding and Initialization

A genetic algorithm works on individuals with chromosomes, which are a representation or codification of the solutions to the problem. In this case, we have chosen an ordinal genetic representation. As shown in Figure 1, the individuals *P1*, *P2*, *O1* and *O2* are identified by sequences so that each element of the

sequence is associated with a numeric identifier that represents a particular job. The population size is set to 40. The initial population is randomly generated except for one individual, which is generated using the NEH Hybrid (NEHH) rule [21], an adaptation of the well known NEH heuristic [19].

3.2 Crossover Operator

The crossover operator generates offspring in general, by coalescing two parents with the objective of generating a better sequence, in this case a better makespan C_{max}. Many crossover operators from the literature are used for the permutation flowshop, such as the Partially Mapped Crossover (PMX), OX, Order Based Crossover (OBX) or Uniform Order Based Crossover (UOBX) [15]. But for regular flowshops and especially the hybrid flowshops these crossover operators give the worst results because they break the building blocks [21]. Introducing dependent setup times and job skipping will complicate the situation even more. The crossover operators that we present aim to ensure a better conservation of the relative order and the absolute order when we deal with dependent setup times. In this work we present three new crossover operators adapted for the SDST/HFFS problem.

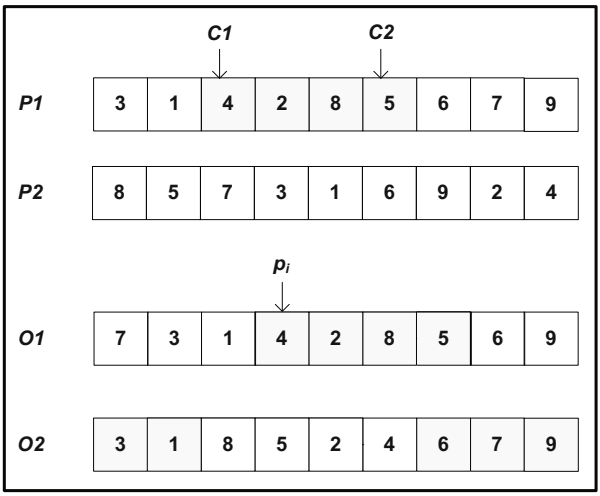

Fig. 1. Illustration of RMPX and ARMPX

The RMPX Crossover. The first crossover operator is the *Random Maximal Preservative Crossover* (RMPX) crossover introduced by [23], which shows good behavior when dealing with the dependent setup times but for a single machine. RMPX is defined as follows : (i) two parents *P1* and *P2* are considered and two

distinct crossover points *C1* and *C2* are selected randomly, as shown in Figure 1; (ii) an insertion point p_i is then randomly chosen in the offspring *O1*, p_i being a random number in the interval [1, n - (*C2* - *C1*)]; (iii) the part [*C1*, *C2*] of *P1*, shaded in Figure 1, is inserted in the offspring *O1* from p_i, from the position 4 as shown in Figure 1; and (iv) the rest of the offspring *O1* is completed from *P2* in the order of appearance from its first position. To generate the second offspring, we just reverse the roles of the two parents P1 and P2 and repeat the same process.

The ARMPX Crossover. The second crossover operator is the *Antagonist Random Maximal Preservative Crossover* (ARMPX), an adaptation of the RMPX crossover where in steps *(ii)* and *(iii)* we insert the parents' first and last parts instead of the part [*C1*, *C2*] in the offspring (shaded in *O2* in Figure 1). After that, the rest of the offspring is completed from the other parent in the order of appearance from its first position. As shown in Figure 1, from parents *P1* and *P2* we obtain the offspring *O2* when applying the ARMPX crossover. The aim here is to preserve the two building blocks in the front and the back of the parent, and to preserve the relative order in the insertion section.

The LJMPX Crossover. The third crossover operator is the *List Jobs Maximal Preservative Crossover* (LJMPX) which works like the RMPX crossover for the first 3 steps. This crossover represent the first contribution in this paper. After inserting the cross section, two lists are built from the second parent *P2* which contains the unscheduled jobs. The LL list contains the jobs which will be inserted to the left of this section while the RL list contains the jobs which will be inserted to the right. An approximate value of the makespan $\overline{C}_{max}$ is then calculated with a sequence containing the subsequence [*C1*, *C2*] in the offspring and where the rest of the jobs have a normalized value $\overline{p}_{ij}$, using a normalized setup time $\overline{s}_{ijk}$ which represents the average processing time and setup time for the unscheduled jobs, respectively. Next, we insert the jobs from the corresponding lists one by one, minimizing the $\overline{C}_{max}$ until we obtain a complete sequence. As shown in Figure 2, the offspring *O3* is a potential offspring from parents *P1* and *P2* where the lists LL and RL are built from the parent *P2* after inserting the part [*C1*, *C2*] from position p_i.

The MPOBX Crossover. The last crossover operator introduced in this paper is the *Maximum Preservative Order Block Crossover* (MPOBX) which works as follows. First, from the two parents, we insert the longest job blocks at the same positions, using four crossover points. After that, as in the LJMPX crossover, we calculate an approximate value $\overline{C}_{max}$ using the $\overline{p}_{ij}$ and $\overline{s}_{ijk}$ for the unscheduled job positions. Then we insert the remaining unscheduled jobs as in the LJMPX crossover, from a single job list. As shown in Figure 3, we insert the block {3, 1, 4} from *P1* and block {9, 7, 2} from *P2*. The unscheduled job list contains jobs 5, 6 and 8. These jobs will be inserted one by one using the $\overline{C}_{max}$ value.

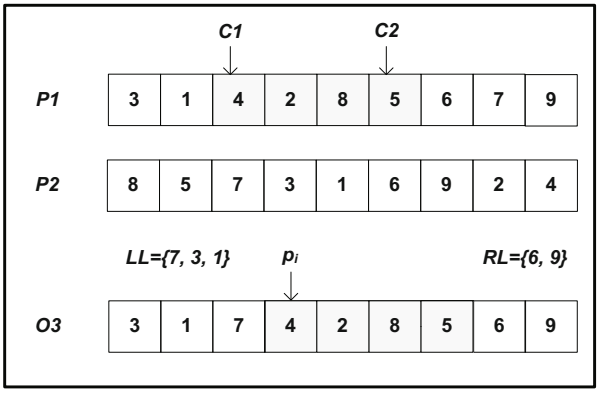

Fig. 2. Illustration of LJMPX

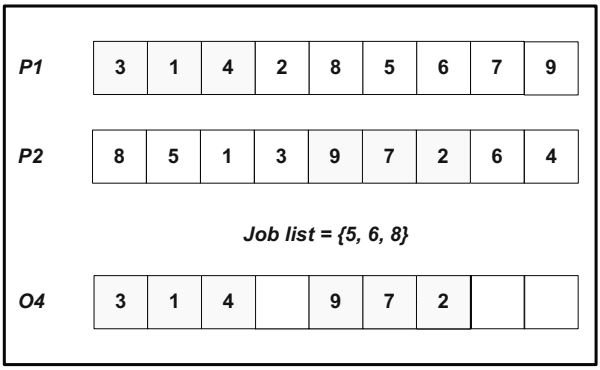

Fig. 3. Illustration of MPOBX

Finally, we set the probability p_c of crossover to 0.8. So, at each generation, $N * 0.8$ offspring will be generated. This crossover represents the second contribution in this paper.

3.3 Mutation Operator, Selection and Replacement Schemes

Mutation consists of exchanging the position of two distinct jobs randomly chosen. The probability p_m of a mutation occurring is set to 0.01. The chromosome selection for the crossover is done using a stochastic binary tournament. The replacement is elitist and uses the $(\lambda + \mu)$ scheme.

4 Ant Colony Optimization for the SDST/HFFS

The ant colony optimization (ACO) is a population based metaheuristic designed to solve combinatorial optimization problems, introduced by [4] and inspired

by studies of the behavior of ants. In the natural world, ants are able to find the shortest path between their nest and food sources, by following a chemical pheromone trail on the ground after they walk on it. That is, they choose the way with more probable paths, which are marked by stronger pheromone concentrations. Indeed, as more ants use the same path, more pheromones are deposited and more ants tend to follow this path. This collective foraging behavior, deposing and following pheromones in the natural world, became the inspiring source of the ACO. [5] proposed notable improvements to the original ACO version. The improvements include a modified transition rule called the pseudo-random-proportional rule, global and local trail updating rules, use of restricted candidates list and the use of local improvement rules. In this section we describe in detail the ACO algorithm to solve the SDST/HFFS problem with the objectives of minimizing the makespan. In the main loop, after the pheromone initialization, has five steps where a ants construct a sequence of N jobs : *(i)* an initial job is set ; *(ii)* each ant builds a job sequence using the pseudo random proportional transition rule in Equations (1) and (2) ; *(iii)* a local pheromone update is performed ; *(iv)* a local improvement heuristic is applied ; and *(v)* a global pheromone update is applied. This main loop is executed until a stopping criterion is reached. The loop is executed for t_{max} cycles, as shown in Figure 4, representing a generic pseudo code embedding the new transition rule shown in Equations (1) and (2). The new transition rule has been adapted to the SDST/HFFS problem. This represents the third contribution in this paper.

4.1 Algorithm Initialization

At each iteration, given that a job i is the previous job added to the sequence, an ant chooses the next job to append by considering, among other factors, the pheromone trail intensity $\tau_{ij}(t)$ which is initialized to a small positive quantity τ_0 for all pair of jobs *(i,j)*, i.e., $\tau_{ij}(0) = \tau_0$. Afterward, the pheromone trail will contain information based on the solution quality and the number of time that ants chose to visit job j after job i.

4.2 Setting Up the Initial Job

Each job has an initial setup time at each stage i on a machine m_i and this setup is taken into account when calculating the makespan. Therefore, to maintain diversity and for each job sequence constructed by the a ants, the first job is chosen pseudo-randomly. This choice is based on the earliest completion time on the first stage.

4.3 Building a Sequence

From an existing partial job sequence, each ant builds a sequence using the pseudo random proportional transition rule in Equations (1) and (2). In Equation (1), q is a random number and q_0 is a parameter; both are between 0 and 1.

```
/* STEP 0 : Pheromone Initialization */
for all job pair (i,j) do
    τ_ij(0) = τ_0
end for
/* Main Loop */
for t = 1 → t_max do
    /* STEP 1 : Set initial job */
    for k = 1 → m do
        Set the initial job for the ant k
    end for
    /* STEP 2 : Build a sequence */
    for i = 2 → n do
        for k = 1 → m do
            Choose the next job to insert using the Equations 1 and 2
            /* STEP 3 : Local pheromone update */
            for all chosen job pair (i,j) do
                τ_ij(t) = p_t * τ_ij(t) + (1 − p_t) * Δτ_ij(t) where Δτ_ij = τ_0
            end for
        end for
    end for
    /* STEP 4 : Local improvement */
    for k = 1 → m do
        Apply local improvement method or-opt heuristic
    end for
    /* STEP 5 : Global pheromone update */
    for all adjacent job pair (i,j) ∈ the best sequence Q* do
        τ_ij(t) = p_t * τ_ij(t) + (1 − p_t) * Δτ_ij(t) where Δτ_ij = 1/L*
    end for
end for
```

Fig. 4. The ACO pseudo-code

The parameter q_0 determines the relative importance of the existing information exploitation and the new solution search space exploration. Indeed, Equation (1) states that the next job will be chosen by a greedy rule when $q \leq q_0$ or by the probabilistic rule of Equation (2) when $q > q_0$. Equation (2) describes the biased exploration rule p_{ij} also adapted to the $((PM)^{(i)})_{i=1}^{m}/F_j, s_{ijk}/C_{max}$ problem when inserting job j after job i.

In these equations, the elements $\tau_{ij}(t)$ and η_{ij} represent the pheromone trail and the visibility, respectively. Concerning the visibility, η_{ij} represents the inverse of the largest completion time among all the jobs in the list of the unselected jobs. Obviously, the completion time includes the setup times between the last scheduled job and the next one. The element $SUCC_{ij}(A_t)$ represents the past information, which is introduced by a matrix built from an archive that stores the best solutions throughout the evolution process, as in some cases in multi-objective evolutionary algorithms using the Pareto-optimal concept. This concept was first introduced in transition rules by [23]; it plays the role of a long-term memory. Here, we adapt them for our problem. From the archive we build a matrix which computes the number of times that a job j follows a job i in the

$$j = \begin{cases} arg \ \ max \left\{ \left[\tau_{ij}(t)\right]^{\alpha} \times \left[SUCC_{ij}(A_t)\right]^{\beta} \times \left[\eta_{ij}\right]^{\delta} \times \left[\frac{1}{\overline{H_{ij}}}\right]^{\phi} \right\} & if \ q \leq q_0 \\ J & if \ q > q_0 \end{cases} \tag{1}$$

where J is chosen according to the probability p_{ij}

$$p_{ij}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha} \times \left[SUCC_{ij}(A_t)\right]^{\beta} \times \left[\eta_{ij}\right]^{\delta} \times \left[\frac{1}{\overline{H_{ij}}}\right]^{\phi}}{\sum \left[\tau_{ij}(t)\right]^{\alpha} \times \left[SUCC_{ij}(A_t)\right]^{\beta} \times \left[\eta_{ij}\right]^{\delta} \times \left[\frac{1}{\overline{H_{ij}}}\right]^{\phi}} \tag{2}$$

archive solutions. Finally, the element H_{ij} represents the lookahead information which use an heuristic that anticipates the choices in the transition rule. This heuristic is based on an upper bound of the makespan, using the average values of processing time $\overline{p_{ij}}$ and the normalized setup times $\overline{s_{ijk}}$ for the unscheduled jobs.

4.4 Local Trail Updating

Once the ants generate a solution, for each pair of jobs *(i,j)* the pheromone level on the path is updated using a local update rule as in Equation (3)

$$\tau_{ij}(t) = p \times \tau_{ij}(t) + (1 - p) \times \Delta\tau_{ij}(t) \tag{3}$$

where $\Delta\tau_{ij} = \tau_0$ and $0 \leq p \leq 1$ is a constant parameter.

4.5 Local Improvement

After computing the makespan of the generated sequence by the ant, we apply a local improvement under probability p_{LI}. For that, we use a simple local search generating a neighborhood using a swap move.

4.6 Global Trail Updating

The pheromone trail is updated at the end of the cycle, but only for the job pairs *(i,j)* in the best solution with makespan C^* found in the cycle. The global update rule is executed using Equation (4)

$$\tau_{ij}(t) = p_t \times \tau_{ij}(t) + (1 - p_t) \times \Delta\tau_{ij}(t) \tag{4}$$

where $\Delta\tau_{ij} = 1/C^*$ and $0 \leq p_t \leq 1$ is a constant parameter.

4.7 Parameter Initialization

The trail pheromone is initialized to the value $\tau_0=(N * L_r)^{-1}$ where N is the job number and L_r is the makespan value of a randomly generated sequence. The other parameters have been assigned the following values : $p = p_t = 0.9$, the ant number $a = 10$ and $q_0 = 0.9$. The parameters α, β, ϕ and δ associated with the four matrices in the transition rule were set to identical values for all the problems. These parameters were adjusted following empirical tests on different instances. The four parameters α, β, ϕ and δ have been assigned the values 4, 2, 3 and 3, respectively. Finally, the archive size and the local improvement probability p_{LI} have been assigned the values 20 and 0.2, respectively.

5 Hybrid Metaheuristic GA/ACO for the SDST/HFFS

We introduce here a collaborative hybridization [20] at the LJMPX crossover introduced in Section 3.2. Indeed, we use the ACO algorithm introduced in Section 4 to fill either the right part or the left part or both. Inserting the jobs on the right of the cross section is similar to the operating of a classical ant. From the last inserted job i in the cross section, a job j is chosen according to the pseudo-random-proportional transition rule expressed in Equations (1) and (2) using the jobs in the right list.

Since the cross section is already set, inserting the remaining jobs on the left of this section can be done either from the first offspring position from left to right as a classical ant or inversely from the first cross section position. During the application of the crossover, we use equiprobably one of the two methods of left insertion. Finally, in the case of inserting jobs from right to left, we make some adaptations in Equations (1) and (2). The hybridization represents the fourth contribution in this paper when using new features in both GA and ACO.

6 Computational Results and Discussion

The benchmark problem set is available from http://soa.iti.es and consists of 960 problem tests. The instances are combinations of N and M, where $N = \{20, 50, 80, 150\}$ and $M = \{2, 4, 8\}$. The processing times are generated from a uniform [1, 99] distribution. The setup times are generated according to four distributions [1, 25], [1, 50], [1, 99] and [1, 125]. This corresponds to a ratio between setup and processing times of 25%, 50%, 100% and 125%, respectively. There is a group with two parallel machines per stage and groups where the number of parallel machines at each stage is sampled from a uniform distribution in the range [1, 4]. The probability of skipping a stage for each job is set at 0.10 and 0.40. All the experiments were run on an Intel Core 2.8 GHz processors and 4 GB of main memory. To evaluate the performance of the other proposed algorithm, we will conduct statistical analysis and comparisons with the results of [17], where, the authors compare an iterated local search (ILS) to several metaheuristics and heuristics as the RKGA of [14], the IA of [28], the genetic

algorithm (GAR) of [21] and the dispatching rules of [13] to cite these methods among others. Many of these compared methods are adapted to the treated problem and recoded for suited comparison purposes. The authors show that the ILS and the GAR represent the best method. So we will compare our algorithm results to these methods. We know that using fixed number of evaluations of the objective function to compare different algorithms allows fair comparisons, but we do not have these information for the reference algorithms. So, we use the same stopping criterion as used in [17], i.e., time computation.

It is important to note that our experiment environment is different from that of [17]. Therefore, we use the following website references [24] and [25] to determine the performance ratio between the two computers. In order to obtain a reliable comparison, all the experiments were done with the stopping criterion set to $n^2 \times m \times 1.5 \times 0.78\, ms$ elapsed CPU time ([17] used $n^2 \times m \times 1.5\, ms$ elapsed CPU time as the stopping criterion for all the compared algorithms). To evaluate the different algorithms we use the performance measure in Equation (5) :

% Increase Over the Best Solution

$$= \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \times 100 \tag{5}$$

where Heu_{sol} is the best makespan obtained by a given algorithm after 10 executions and $Best_{sol}$ is the best known makespan.

First, we produce experiments in order to compare crossover operators embedded in the GA independently : OX, PMX, UOBX [15], S2BOX, used in the GAR [21], RMPX [23], ARMPX, LJMPX and MPOBX. Each crossover operator is embedded independently in the GA presented in Section 3 with 500 generations as the stopping criterion. The result summaries are presented in Table 1. Indeed, the results represent the group instances average, and the best averages are in boldface type.

As shown in this table, the *List Jobs Maximal Preservative Crossover* (LJMPX) and the *Maximum Preservative Order Block Crossover* (MPOBX) present the best results among the eight tested, and their results are very similar except for the large instances where LJMPX allows for achieving a slightly better average. This supports the idea that these two crossover operators are more adapted to the studied problem. Indeed, using the approximate value of the makespan when fulfilling the unscheduled jobs in the sequence gives the two crossover operators more accuracy when dealing with more stages. Also, using the list jobs allows us to better optimize the setup times when choosing jobs to insert.

Furthermore, maintaining the blocks of jobs in the crossover operators improves the performance of the algorithms, i.e., the LJMPX maintains the cross section while the MPOBX maintains blocks from the two parents. Hence, both crossover operators take greater account of the relative and absolute position of jobs when maintaining blocks. Finally, as shown in Table 1, MPOBX allows us to achieve a slightly better average than LJMPX, particularly for larger instances. This can be explained by the fact that MPOBX conserve more absolute positions when maintaining blocks from parents. Moreover, LJMPX compensates this behavior by

Table 1. Comparison of different crossovers (% Increase over the best solution)

Instances	OX	PMX	UOBX	S2BOX	RMPX	ARMPX	LJMPX	MPOBX
20*2	7.82	7.27	7.43	6.93	5.13	4.39	3.78	**3.39**
20*4	5.25	8.25	8.15	3.89	5.65	5.24	2.80	**2.27**
20*8	6.87	6.40	8.44	3.09	5.71	6.31	2.85	**2.49**
50*2	8.36	6.23	9.24	9.01	5.82	4.82	3.81	**3.26**
50*4	7.33	7.39	9.53	5.85	6.69	4.24	2.74	**1.85**
50*8	9.35	12.24	8.97	3.79	6.67	4.28	**3.73**	3.75
80*2	5.29	9.31	7.84	8.28	5.64	4.19	2.90	**2.06**
80*4	8.23	8.39	9.49	7.39	6.63	5.70	**2.74**	2.84
80*8	10.27	10.43	10.19	6.53	6.56	5.95	3.78	**3.48**
120*2	8.05	7.27	9.19	10.58	7.65	7.45	**3.87**	4.02
120*4	9.30	10.39	8.97	9.04	6.62	8.23	**5.95**	6.02
120*8	10.31	13.13	10.82	10.59	8.78	9.82	**6.81**	7.21
Average	**8.04**	**8.89**	**9.02**	**7.08**	**6.46**	**5.89**	**3.81**	**3.55**

providing more exploration when inserting the cross section but conserving the relative order and consequently conserving the setup times between jobs especially when the setup times have a significant impact when calculating the makespan. This mechanism allows LJMPX to achieve a slightly better average for the large instances where more exploration is needed to obtain better results.

We proceed now with the comparisons of the proposed algorithms (GA, ACO and HGA) against the ILS of [17] and the GAR of [21]. These two algorithms have shown high performance in the original papers in which they were proposed [17,21]. The authors compare these algorithms with four high performing algorithms : two genetic algorithms, immune algorithm and ant colony optimization. To sum up, the ILS and GAR algorithms showed the best results in their respective studies. The GA version retained here is the one embedding the two crossover operators LJMPX and MPOBX. The choice of the crossover to apply is made randomly, i.e., by a fair coin toss. This policy was chosen as result of computational experiments. The results are presented in Table 2; they also represent the group instances averages. The best averages are also in boldface type.

The first observation is that the new GA and ACO algorithms always provide a better average than the ILS and the GAR algorithms. Also, if we compare the GA and ACO algorithms, the first provides a better average on all the group instances except for the 20× 2, 50× 2, 80× 2 and the 120× 2 group instances where the ACO does slightly better. These group instances are those with 2 stages. It seems that the transition rule embedded in the ACO algorithm performs better in these configurations. In general, the GA has a better average than the ACO, with 0.60 and 0.76, respectively. We can also remark that there is a non negligible improvement in comparison with the ILS and the GAR averages.

Now, if we focus on the HGA algorithm results, we can see that this algorithm provides the better average for all the group instances except for the 20× 2 group

Table 2. Comparison of the ILS [17], the GAR [21], the ACO, the GA and the HGA (% Increase over the best solution)

Instances	GA	ACO	HGA	ILS	GAR
20*2	0.75	**0.49**	0.51	1.70	3.82
20*4	0.51	0.69	**0.49**	1.90	3.90
20*8	0.43	0.75	**0.40**	1.70	4.02
50*2	0.69	0.61	**0.55**	2.72	4.65
50*4	0.52	0.90	**0.36**	2.98	4.73
50*8	0.66	1.12	**0.45**	3.48	5.01
80*2	0.59	0.58	**0.45**	3.29	5.29
80*4	0.48	0.74	**0.44**	2.01	4.87
80*8	0.71	1.04	**0.51**	4.87	6.03
120*2	0.58	0.57	**0.54**	3.23	5.32
120*4	0.59	0.79	**0.51**	4.36	5.05
120*8	0.67	0.89	**0.53**	5.74	7.02
Average	**0.60**	**0.76**	**0.48**	**3.16**	**4.98**
Median	**0.53**	**0.71**	**0.36**	**2.45**	**4.81**
Std	**0.53**	**0.55**	**0.48**	**2.46**	**2.39**

where the ACO obtains a better average. Also, in the same vein, the HGA significantly improves all the average results in comparison to the ILS and the GAR algorithms. Furthermore, combining both the GA and ACO mechanisms in the HGA marginally enhances the results in comparison of the GA and the ACO algorithms. If we observe the standard deviation values, we remark that those of the proposed methods (ACO, GA and HGA) are very low. This can be explained by the effect of both mechanism used in the ACO and the GA, which use an upper bound and consequently smooth over the results.

Moreover, to significantly compare the proposed algorithms, we conducted a pairwise comparison to detect significant performance differences between all the algorithms with the non-parametric Wilcoxon signed-rank test [27] for each instance using the results of our 10 runs with an error probability of 1% over the numerical results. We remind the reader here, that the Wilcoxon test does not require assumptions regarding the distribution results. Indeed, for the purpose of a pairwise heuristics comparison, the Wilcoxon test assumes that the first heuristic median M1 equals the second heuristic median M2 hypothesis is null and that M1 $\neq$ M2 is the alternative hypothesis.

The Wilcoxon test results are shown in Table 3 where bold values indicate where the null hypothesis is rejected. The critical values for all tests are identical and between -2.575 and 2.757. Thus, the Wilcoxon test indicates with a confidence level of 99% that the HGA algorithm statistically outperforms all other methods. Moreover, GA statistically outperforms ACO, ILS and GAR. Further more, ACO surpasses ILS and GAR. Finally, regarding the quality solutions, we obtain the following ranking : HGA-GA-ACO-ILS-GAR.

Table 3. Wilcoxon test *t-value* for the ILS [17], the GAR [21], the ACO, the GA and the HGA

	HGA	GA	ACO	ILS	GAR
HGA	--	-4.29	-10.4	-25.78	-26.71
GA		--	-6.1	-25.4	-26.7
ACO			--	-24.39	-26.66
ILS				--	-16.27
GAR					--

7 Conclusion

In this work, we have introduced three new algorithms : a genetic algorithm (GA) embedding two new crossovers, an ant colony optimization algorithm (ACO) that integrates lookahead information and archive concepts in the transition rule and a hybrid genetic algorithm (HGA) integrating the ACO in the GA crossover to minimize the makespan in a hybrid flexible flowshop with sequence-dependent setup times. The proposed approaches are essentially based on adapting different algorithm mechanisms to the specificities of the studied problem, *i.e* the crossovers in the GA and the transition rule in the ACO. Indeed, after inserting the cross section from the first parent, the two crossover operators use heuristics and lists from the second parent to insert the remaining jobs. These heuristics are replaced by the ACO algorithm in the HGA algorithm. For its part, the pseudo random proportional transition rule embedded into the ACO integrates past, present and future information to build a sequence. The numerical experiments allowed us to demonstrate the efficiency of our approaches to this problem.

Our results encourage us to use such approaches, with hybridization, for other scheduling problems in particular and other optimization problems in general. It is in this direction that our work will be directed.

References

1. Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M.Y.: A survey of scheduling problems with setup times or costs. European Journal of Operational Research **187**(3), 985–1032 (2008)
2. Allahverdi, A., Soroush, H.: The significance of reducing setup times/setup costs. European Journal of Operational Research **187**(3), 978–984 (2008)
3. Conner, G.: 10 questions. Manufacturing Engineering Magazine, pp. 93–99 (2009)
4. Dorigo, M.: Optimization, Learning and Natural Algorithms. Ph.D. thesis, Politecnico di Milano, Italy (1992)
5. Dorigo, M., Gambardella, L.: Ant colony system: A cooperative learning approach to the traveling salesman problem. In: IEEE Transactions on Evolutionary Computation (1997)
6. Dudek, R., Smith, M., Panwalkar, S.: Use of a case study in sequencing/scheduling research. Omega **2**(2), 253–261 (1974)

7. Gomez-Gasquet, P., Andres, C., Lario, F.: An agent-based genetic algorithm for hybrid flowshops with sequence dependent setup times to minimise makespan. Expert Systems with Applications **39**(9), 8095–8107 (2012)
8. Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.G.H.R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics **5**, 287–326 (1979)
9. Gupta, J.N.D.: Two-stage, hybrid flowshop scheduling problem. The Journal of Operational Research Society **39**(4), 359–364 (1988)
10. Holland, J.H.: Adaptation in natural and Artificial Systems. MIT Press, Cambridge, MA, USA (1992)
11. Jabbarizadeh, F., Zandieh, M., Talebi, D.: Hybrid flexible flowshops with sequence-dependent setup times and machine availability constraints. Comput. Ind. Eng. **57**(3), 949–957 (2009)
12. Javadian, N., Fattahi, P., Farahmand-Mehr, M., Amiri-Aref, M., Kazemi, M.: An immune algorithm for hybrid flow shop scheduling problem with time lags and sequence-dependent setup times. The International Journal of Advanced Manufacturing Technology **63**, 337–348 (2012)
13. Kurz, M.E., Askin, R.G.: Comparing scheduling rules for flexible flow lines. International Journal of Production Economics **85**(3), 371–388 (2003)
14. Kurz, M.E., Askn, R.G.: Scheduling flexible flow lines with sequence-dependent setup times. European Journal of Operational Research **159**(1), 66–82 (2004)
15. Michalewicz, Z.: Genetic algorithms + data structures = evolution programs, 3rd edn. Springer-Verlag, London (1996)
16. Mirsanei, H., Zandieh, M., Moayed, M., Khabbazi, M.: A simulated annealing algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times. Journal of Intelligent Manufacturing **22**, 965–978 (2011)
17. Naderi, B., Ruiz, R., Zandieh, M.: Algorithms for a realistic variant of flowshop scheduling. Comput. Oper. Res. **37**(2), 236–246 (2010)
18. Naderi, B., Zandieh, M., Roshanaei, V.: Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum tardiness. The Internayional Journal of Advanced Manufacturing Technology **41**, 1186–1198 (2009)
19. Nawaz, M., Enscore, E.E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega **11**(1), 91–95 (1983)
20. Puchinger, J., Raidl, G.R.: Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification. In: Mira, J., Álvarez, J.R. (eds.) IWINAC 2005. LNCS, vol. 3562, pp. 41–53. Springer, Heidelberg (2005)
21. Ruiz, R., Maroto, C.: A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European Journal of Operational Research **169**(3), 781–800 (2006)
22. Ruiz, R., Stützle, T.: An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European Journal of Operational Research **187**(3), 1143–1159 (2008)
23. Sioud, A., Gravel, M., Gagné, M.: A hybrid genetic algorithm for the single machine scheduling problem with sequence-dependent setup times. Computers & OR **39**(10), 2415–2424 (2012)
24. SPEC: Intel core 2 duo p8400 @ONLINE (2009). http://www.spec.org/cpu2006/results/res2010q1/cpu2006-20100118-09344.html
25. SPEC: Intel core i7–2600 @ONLINE (2011). http://www.spec.org/cpu2006/results/res2011q3/cpu2006-20110718-17542.html

26. Vignier, A., Billaut, J.C., Proust, C.: Les problemes d'ordonnancement de type flow-shop hybride : etat de l'art. RAIRO - Operations Research - Recherche Operationnelle **33**(2), 117–183 (1999)
27. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics **1**, 80–83 (1945)
28. Zandieh, M., Fatemi Ghomi, S., Moattar Husseini, S.: An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. Applied Mathematics and Computation **180**(1), 111–127 (2006)
29. Zhu, X., Wilhelm, W.E.: Scheduling and lot sizing with sequence-dependent setup: A literature review. IIE Transactions **38**(11), 987–1007 (2006)

http://www.springer.com/978-3-319-12969-3

Swarm Intelligence Based Optimization
First International Conference, ICSIBO 2014, Mulhouse, France, May 13-14, 2014. Revised Selected Papers
Siarry, P.; Idoumghar, L.; Lepagnot, J. (Eds.)
2014, X, 193 p. 55 illus., Softcover
ISBN: 978-3-319-12969-3