7e Conférence Internationale de MOdélisation et SIMulation - MOSIM'08 - du 31 mars au 2 avril 2008 – Paris- France

« Modélisation, Optimisation et Simulation des Systèmes : Communication, Coopération et Coordination.».

METHOD BASED ON ANT COLONY SYSTEM FOR SOLVING THE HYBRID FLOW SHOP SCHEDULING PROBLEM

S. Khalouli, F. Ghedjati, A. Hamzaoui

CReSTIC-Reims
UFR Sciences Exactes et Naturelles
Moulin de la Housse, BP 1039, 51687 Reims cedex 2
safa. khalouli@etudiant.univ-reims.fr,
{fatima.ghedjati, abdelaziz.hamzaoui}@univ-reims.fr

ABSTRACT : *This paper proposes an adaptation of the ant colony system (ACS) algorithm to solve the hybrid flow shop (HFS) scheduling problems. The objective is to determine a schedule that minimizes the total completion time. This problem has been shown NP-hard. Thus the approximate methods are more suitable for this kind of problems in comparison with the exact ones. Our approximate method consists on solving the problem by phases. The first phase consists in some static heuristics to deal with the assignment problem. The second phase is dedicated to the sequencing problem. This latter sub-problem is solved by using an adaptation of the ACS algorithm. In order to improve the ACS algorithm a local search is added. According to the importance of the ACS parameters in the quality of the solution, some preliminary tests are realized in order to find the best operating parameters. The algorithm is tested by using benchmark problems taken from the literature and randomly generated. It is found after extensive computational investigation that the proposed ant colony algorithm gives promising and good results.*

KEYWORDS : *scheduling, hybrid flow shop, ant colony optimization, ant colony system, meta-heuristic, local search.*

1. INTRODUCTION

In the last decade, scheduling in a hybrid (HFS) or multi-processor flow shop has received attention because of its importance from both theoretical and practical points of view. This problem has been showed as an adequate model for the study of a great number of production systems. Thus, it is widely used in process industries.

The manufacturing environment of the hybrid flow shop is considered as an extension of the classical flow shop. In fact, it presents a multistage production process with the property that all the jobs have to pass through a number of stages in the same order with several parallel machines at each stage. The duplication of the number of machines in some stages introduces additional flexibility to the production process. Hybrid flow shop models differ in the type of machines at the stages. The machines may be identical, uniform or unrelated (Blazewicz *et al.*, 1996). In this paper, parallel machines are assumed to be identical.

To solve the HFS problem, we have to determine both an assignment and a sequencing problem, according to a given objective. Here, the objective is to find a solution that minimizes the maximum completion time of the jobs or makespan, denoted $C_{\max}$. This problem was shown as *NP*-hard in the strong sense for the simplest case, made up of two stages and having at least two machines available in one of the stages (Gupta, 1988). In order to solve HFS problems, exact methods can be used to find optimal solutions of only small-sized problems. Generally, these methods perform very poorly in large-sized problems due to the excessive computing time. For practical purposes, it is more appropriate to use approximate methods. They are able to achieve good solutions for scheduling problem in an acceptable time. The literature has proposed many solution methodologies for the hybrid flow shop. Among these methods, we quote hereafter some of them: genetic algorithms (Portmann *et al.*, 1998) and (Jin *et al.*, 2002), tabu search approach (Nowicki and Smutnicki, 1998), simulated annealing (Gourgand *et al.*, 2000) and (Jin *et al.*, 2006), approach based on artificial immunized system (AIS) (Engin and Doyen, 2004) and ant algorithms (Alaykýran *et al.*, 2007).

The purpose of this paper is to present an ant algorithm based on the ant colony system (ACS) to resolve the HFS problem. The remainder of this paper is organized as follows. In the next section, we give a formal description of the HFS problem. In section 3, a brief presentation of the ant colony optimization algorithms is proposed. In section 4, we introduce the proposed algorithm. Computational results are provided in section 5. Finally, conclusion and further research direction are given.

2. PROBLEM NOTATION

The hybrid flow shop is a generalization of the traditional flow shop in which there are a set of stages in series. At each stage s $(s = 1, 2, \ldots, S)$ there is m_s $(m_s \geq 1)$ identical

parallel machines, denoted $M_s = (M_{s1}, M_{s2}, \ldots, M_{s,m_s})$. The machines are assumed available from time zero. A set $J = \{1, 2, \ldots, n\}$ of *n* jobs has to be successively processed in the *S* stages. Each job *j* needs several operations $(o_{1j}, o_{2j}, \ldots, o_{Sj})$. An operation o_{ij} can be processed on any of the machines at stage *i* during p_{ij} time units and can start only after all its preceding ones $o_{1,j}, \ldots, o_{i-1,j}$ have been completed. A machine can only perform one operation at a time and preemption is not allowed. Solving this problem consists of assigning operations to machines on each stage (routing problem) and sequencing the operations assigned to the same machine (scheduling problem). The goal is to minimise the makespan, *i.e* the completion time of the last job at the last stage denoted by $C_{\max} = \max_{1 \le j \le n}\{C_{Sj}\}$.

According to the well-known field notation $\alpha / \beta / \gamma$ (Graham *et al.*, 1979) for scheduling problems and the extension proposed by (Vignier *et al.*, 1999), the HFS problem can be noted as $FHS, ((Pm_s)_{s=1}^{S}) \mid C_{\max}$.

Example1. Consider the $FH\,3, (P1, P\,2, P\,2) \mid C_{\max}$ problem where three jobs have to be processed through three stages. The number of machines in each stage is respectively $m_1 = 1, m_2 = 2$ and $m_3 = 2$. The processing times are as follows: $p_{11} = 4$, $p_{21} = 6$, $p_{31} = 10$, $p_{12} = 2$, $p_{22} = 8$, $p_{32} = 2$, $p_{13} = 4$, $p_{23} = 4$ and $p_{33} = 4$. The makespan is equal to 20.

3. ANT ALGORITHMS

The ant colony optimization (ACO) algorithms form a class of meta-heuristic based on a natural phenomenon: the behaviour of ants in their cohabitation in colonies. In fact, some observations have shown that ants, although they individually have limited capacities, can find the shortest path from a food source to their nest without visual cue. They are also capable of adapting to changes in the environment. Thus, the appearance of an unexpected obstacle on the initial path between the food source and the nest doesn't stop the ants from finding another path, the shortest possible. To cooperate and to communicate with each other, the ants adopt chemical volatile substance called pheromone. Concretely, the ants deposed some pheromone on the ground as they move about. Being very sensitive to this substance, an ant chooses in a randomly way the path comprising a strong concentration of this substance. Thus, when several ants cross the same space, an emergence of the shortest path is obtained. Based on this process, the ACO is an iterative method able of finding the shortest path connecting sources on a weighted graph which represents the optimization problem. It has been introduced for the first time by (Colorni *et al.*, 1991) to solve the travelling salesman problem (TSP). It uses a multi-agent system formed by several artificial ants. An artificial ant is different from a natural counterpart by the following characteristics:

- It (i.e an artificial ant) is not completely blind. Indeed, it uses a value called visibility or heuristic value which finds information about the specific problem. This value can influence the choice of the ant and guided it differently;
- It has a memory which is used to store the search history;
- It can manage the deposited quantity of pheromone according to the quality of the solution; moreover it is possible to have various types of pheromones.

The ACO meta-heuristics are evolutionary methods. Unlike genetic algorithms, they have the feature to seek solutions basing on the totality of the population and not only on some best individuals. Thus, each ant uses the collective experience to find a solution to the problem. A solution is built step by step and the choice made by an ant depends on the quantity of pheromone and the heuristic value.

The ACO meta-heuristics have been successfully used for solving a range of combinatorial optimization problems such as, travelling salesman problem, the vehicle routing problem, the quadratic assignment problem and the scheduling problem. The first ACO algorithm is called ant system (AS) (Colorni *et al.*, 1991). Then, AS was improved and extended. The improved versions include the ant colony system (ACS) (Dorigo and Gambardella, 1997) and the MAX-MIN ant system (MMAS) (Stützle and Hoos, 1997), etc. Several studies have applied ACO algorithms to solve different type of scheduling problems such as:

- the single machine scheduling problem (Den Besten *et al.*, 2000), (Gagné *et al.*, 2002) and (Ying and Liao, 2003);
- flow shop (Stützle, 1998), (Ying and Liao, 2004), (Gajpal *et al.*, 2006) and (Ying and Lin, 2006);
- job shop (Colorni *et al.*, 1994) and (Zhang *et al.*, 2006);
- parallel machines shop problem (Sankar *et al.*, 2005);
- flexible scheduling problem (Nait Tahar *et al.*, 2005) and (Alaykýran *et al*., 2007).

4. OUR PROPOSED METHOD : THE IMPROVED ACS FOR THE HFS

The suggested method consists on solving the problem by phases. The first phase consists in some static heuristics to deal with the assignment problem. The second phase is dedicated to the sequencing problem. This latter subproblem is solved by adapting the ACS algorithm

4.1. Assignment problem

To each repartition of the operations on the machine set, we associate an assignment. Then, each operation has to be assigned to a machine according to a specific technique of the scheduling generator. Indeed, if there are multiple

choices (Ghedjati and Pomerol, 2000), the assignment of the operations can be done by static or dynamic rules. In case of static rules, the assignment is realized before the scheduling process and thus remains invariant along the process. In the other case, the rules use an instantaneous knowledge of the system. As a consequence, they determine the operations priorities during the process of scheduling. Each assignment is characterized by a set:

$A = \{A_{i,j,k} \in \{0,1\} \mid 1 \le i \le S, 1 \le j \le n, 1 \le k \le m_i\}$ where

$A_{i,j,k} = \begin{cases} 1 & \text{if } o_{ij} \text{ is assigned to } M_k \\ 0 & \text{otherwise} \end{cases}$ and $\sum_{k=1}^{m_i} A_{i,j,k} = 1$.

To solve the assignment problem, we propose some static heuristics based on some dispatching rules. These heuristics determine a job-sequence for the first stage according to a dispatching rule. Then, jobs are assigned to the machines at the first stage by using the first available machine (FAM) rule. For the other stages ($1 < s \le S$), the first in first out (FIFO) rule is used to find the next job sequence by means of the job sequence of the previous stage. Afterwards, the jobs are assigned to the machines at the remained stages by applying the FAM rule. The used dispatching rules are: the shortest processing time (SPT), the longest processing time (LPT), the least work remaining (LWKR), the most work remaining (MWKR), the shortest remaining processing time (SRT) and the longest remaining processing time (LRT) (Baker, 1974).

4.2. Scheduling (sequencing) Problem

There are a number of design decisions to make when developing an ACO algorithm to tackle a combinatorial optimization problem (Blum and Sampels, 2002). The first issue is to represent this problem with a graph model solvable by the ACS approach. Then, as described in section 3, the pheromone and the heuristic value are two primordial components which guide move selection and, as a consequence, they influence system performance. For these reasons, we have to choose an adequate pheromone model representation and a heuristic value which can find information about the specific problem. Another important point, in order to achieve feasible solutions, is to respect the precedence constraints of the problem at each construction step. This is ensured by using a restricted list (called "candidate list") of all the operations that can be selected. In what follows, we expose these components in details.

4.2.1 Disjunctive graph

The key to apply the ACS to a new problem is to identify an appropriate representation for the problem. According to (Negenman, 2001), the HFS problem can be represented by a disjunctive graph $G = (O, C, D)$ (see Figure 1) defined as follow:

- O is the node set, consisting of all the operations o_{ij}. Two fictive nodes (o_B^* and o_E^*) are added to mark the beginning and the end of each job j.
- C is the set of conjunctive (directed) arcs representing the precedence relationships between the operations of the same job.
- D is the set of disjunctive (undirected) edges; each pair of operations in a stage s is related by an edge.

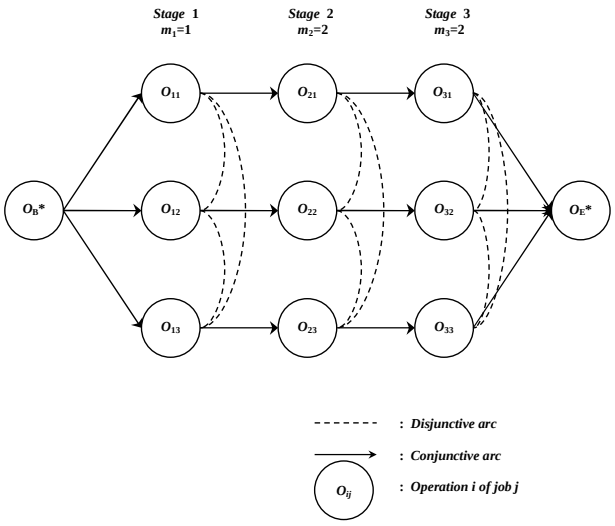

Figure 1. Disjunctive graph for the HFS problem of Example 1.

A feasible schedule is obtained by turning the edges of mutually disjoint cliques into arcs. In case of single machine, the resulting graph contains only one clique per stage, while the multiprocessor flow shop may contain more than one clique at each stage.

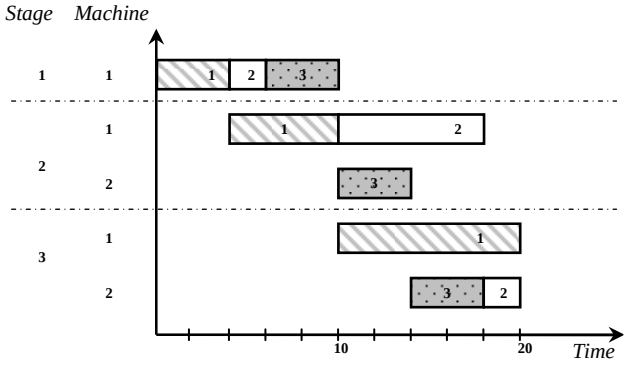

Figure 2. The Gantt chart for the instance of Example 1.

A feasible solution to Example1 can be represented as follow: for the only available machine in the first stage, we schedule the operations in this order o_{11}, o_{12}, o_{13}. In the second stage, operations o_{21}, o_{22} are assigned to the same

machine. o_{21} is treated before o_{22}. Operation o_{23} is treated on the other machine. In the last stage, the operations o_{33}, o_{32} are respectively assigned to the same machine. o_{31} is assigned to the other machine. The Gantt chart of the solution is shown in Figure 2.

4.2.2 Pheromone representation and visibility function

When building a solution, an ant selects the next operation according to the amount of pheromone on arcs and the visibility function associated with the operations connected by arcs. Then, the two components have an impact in the solution performance.

***Pheromone model*:** Many pheromone models, for scheduling problem, are based on assigning a pheromone value $\tau(o_{ij}, o_{hl})$ on every pair of operations $o_{ij}, o_{hl} \in O$ as shown in (Colorni *et al*, 1994). In this last representation, the pheromone value $\tau(o_{ij}, o_{hl})$ bias the choosing of an ant traversing two sequenced operations o_{ij} and o_{hl}, regardless the machines to which they are assigned. Other pheromone representations have been presented in literature (Blum and Sampels, 2002). In the "relation-learning model" representation, pheromone values are assigned to pairs of related operations: in this case two operations $o_{ij}, o_{hl} \in O$ are related if they have to be processed on the same machine resource. Thus, pheromone influences the relative order of operations requiring the same machine.

As the aim of our problem is to sequencing a set of operations through a set of machines, we try to influence ants building a solution in ordering operations treated on a same machine by combining the two models exposed above. The proposed pheromone model consists on assigning a pheromone value $\tau(o_{ij}, o_{hl})$ on every pair of operations, but these values are different according to the machines to which they are assigned. In fact, in our representation, the quantity of pheromone assigned to pairs of related operations is set more important than those not related. An initial quantity of pheromone is assigned to every arc connecting a pair of operations. This initial pheromone τ_0 level is assumed to be more important if operations are related.

***Visibility function*:** The visibility $\eta(o_{ij}, o_{hl})$ finds information about the specific problem. It is used to estimate the desirability of transition from an operation o_{ij} to another operation o_{hl}. In case of scheduling problem, visibility function is different to the reciprocal distance from a node to another as proposed in the ACS algorithm for the TSP problem (Dorigo and Gambardella, 1997). It can be used to represent information related to the partial sequencing at a time. In order to bias the transitions probabilities many dispatching rules have been used in the literatures (Blum and Sampels, 2004) and (Ying and Lin, 2006). Dispatching rules are policies used to select an operation from the set of operations that may be scheduled next. They are constructive heuristics which construct schedule based on simple rule. In this paper, we use the following 8 different heuristics based on the following dispatching rules: SPT, LPT, LWKR, MWKR, SRT, LRT, EST (the Earliest Starting Time) and EFT (the Earliest Finishing Time) (Blum and Sampels, 2004).

4.2.3 Construction procedure

Initialize the pheromone trail for every edge;
Set the operating parameters.

Loop */*at this level each loop is called an iteration*/*

A colony of ants is initially positioned on the starting node.

Loop */*at this level each loop is called a step*/*

Loop */*at this level each loop is called a transition*/*

Apply for each ant, the transition rule to select the next processing operation.

Apply the local updating rule to decrease pheromone on visited edge.

Until a complete tour is realized

Until all ants have finished complete schedule

Apply local search approach to improve the best schedule (not applied at each iteration).

Apply the global updating rule to increase pheromone on edges of the current best schedule and decrease pheromone on other edges.

Until satisfying stopping criterion.

Our ACS proposed Algorithm

To construct a feasible solution, each ant builds independently a sequence of operations by performing the construction step. From an existing operation sequence o_{ij}, an ant *k* must choose from the "candidate list" L_k (which is the set of feasible successor operations that remain to be visited by ant *k*), the next operation y to append by using a pseudo-random rule, based on the pheromone trails τ and the heuristic value η. This rule is formally given by equations (1) and (2) (Dorigo and Gambardella, 1997):

$$y = \begin{cases} \arg \max_{o_{hl} \in L_k(o_{ij})} \left\{ \left[\tau(o_{ij}, o_{hl}) \right] \cdot \left[\eta(o_{ij}, o_{hl}) \right]^{\beta} \right\}, & \text{if } q \le q_0 \\ Y, & \text{if } q > q_0 \end{cases} \quad (1)$$

where β is a parameter which determine the relative importance between $\tau(o_{ij}, o_{hl})$ and $\eta(o_{ij}, o_{hl})$; q is a random number uniformly distributed in [0,1] and q_0 $(0 \le q_0 \le 1)$ is a parameter that determines the relative importance between exploitation and exploration. Indeed, the system

tends to carry out a diversification if $q > q_0$. Otherwise, the system tends towards intensification and consequently the algorithm exploits more information, collected by the system. Y is a random variable selected according to the random-proportional rule (Dorigo and Gambardella, 1997):

$$p_k(o_{ij}, Y) = \frac{\left[\tau(o_{ij}, Y)\right].\left[\eta(o_{ij}, Y)\right]^{\beta}}{\sum_{o_{hl} \in L_k(o_{ij})} \left[\tau(o_{ij}, o_{hl})\right].\left[\eta(o_{ij}, o_{hl})\right]^{\beta}} . \quad (2)$$

The procedure is repeated until all the operations are selected. The selected operations are successively stored into a "*tabu list*" which is used to record the visited nodes of the current schedule. It is noted that the "*tabu list*" is different from the tabu search method and its goal is to avoid an operation to be visited twice.

4.2.4 Pheromone updating

The mechanism of updating pheromone is used to simulate the changes in the amount of pheromone. In fact, this latter is augmented due to the new pheromone deposit by ants and diminished according to the pheromone evaporation. Two kinds of pheromone updating strategies are proposed in (Dorigo and Gambardella, 1997).

***Local pheromone updating rule*:** In order not to influence the choice of the other ants, the local updating rule reduces the pheromone level of the visited edges by an ant *k*. Thus, these edges become less attractive for the other ants. Indeed, this rule is used to shuffle the tour of the other ants and avoid local optima. When an ant *k* selects an operation, a local updating rule is made to modify the quantity of pheromone to its visited edges according to equation (3):

$$\tau(o_{ij}, o_{hl}) = (1 - \rho_\ell).\tau(o_{ij}, o_{hl}) + \rho_\ell.\tau_0 . \quad (3)$$

where τ_0 is the initial pheromone level and ρ_ℓ $(0 \le \rho_\ell \le 1)$ is the pheromone evaporating parameter.

***Global pheromone updating rule*:** The pheromone trail is also updated at the end of each iteration. Only the best solution is globally updating. The pheromone global updating rule is defined as follow:

$$\tau(o_{ij}, o_{hl}) = (1 - \rho_g).\tau(o_{ij}, o_{hl}) + \rho_g.\Delta\tau(o_{ij}, o_{hl}) . \quad (4)$$

where

$$\Delta\tau(o_{ij}, o_{hl}) = \begin{cases} 1/C_{gb} & \text{si } (o_{ij}, o_{hl}) \in \text{global best tour} \\ 0 & \text{otherwise} \end{cases} .$$

In the above equation, C_{gb} is the length of the best solution and ρ_g $(0 \le \rho_g \le 1)$ is the pheromone evaporating parameter of the global updating rule. According to these equations, global updating rule is applied to intensify pheromone levels on the edges belonging to the best tour.

4.2.5 Local search approach

To improve the obtained solution, a local search strategy is included to our approach. The proposed local search strategy is based on the neighbourhood structure proposed by (Nowicki and Smutnicki, 1998). This procedure can enable the reassignment of operations to other machines and finding improving solutions. We propose to apply this approach with a certain probability (P_{LS}). When the local search is applied, the new assignment (if there exist) of the operations to the machine resources is used for the next iterations. We can conclude that the local search allows the modification of the proposed static assignment during the algorithm.

5. EXPERIMENTATIONS

In this section we present the results of a series of computational experiments conducted to test the effectiveness of the proposed approach. Henceforth, we call our algorithm *ACS-HFS*. The instances problem are taken from the literature and randomly generated.

In order to test these problems with our proposed *ACS-HFS* algorithm, we have to determine all the parameters of this latter. Indeed, these parameters have a great impact in the quality of the solution (Dorigo and Gambardella, 1997). These parameters are: $q_0, \beta, \rho_\ell, \rho_g, \tau_0$, the number of the ants and the number of the iterations. Some preliminary tests have been conducted to find the parameter settings $q_0, \beta, \rho_\ell, \rho_g$. The best parameters values are reported in Table 1.

Parameter	Range	Best Value
q_0	$0 \le q_0 \le 1$	0,7
β	$0 \le \beta \le 10$	2
ρ_ℓ	$0 \le \rho_\ell \le 1$	0,5
ρ_g	$0 \le \rho_g \le 1$	0,5

Table 1. Values of the used parameters in the *ACS-HFS* algorithm.

For each instance, the number of ants is assumed to be equal to the number of jobs. The number of iterations is assumed to be equal to 2000 as in (Alaykýran *et al*., 2007). Then, the algorithm terminates when either the lower bound (*LB*) or the 2000 iterations are reached. The initial pheromone level τ_0 is set equal to $(n \times S \times LB)^{-1}$. For all arcs connecting two operations we set τ_0, except those connecting related operations to the same machine. For these latter, we assume that the initial pheromone level is equal to $(5 \times \tau_0)$. We apply the local search with the probability P_{LS} equal to $0,3$.

Comparisons with the *LB* are conducted to evaluate the deviation of best *ACS-HFS* version, from the optimal solution. The percentage deviations (%Deviation) are computed according to the following equation:

$$\%\text{Deviation} = \frac{\text{Best}C_{\max} - LB}{LB} \times 100 . \quad (5)$$

The platform of our experiments is a personal computer with a Pentium (R) 4, 2.66 GHz, and 448 Mo of RAM. Our algorithms are coded in Java.

5.1. Literature benchmarks

In this part, computational tests were performed on 77 benchmark problems taken from (Carlier and Néron, 2000). The problem sizes are respectively: $10\text{jobs} \times 5\text{stages}$, $10\text{jobs} \times 10\text{stages}$, $15\text{jobs} \times 5\text{stages}$ and $15\text{jobs} \times 10\text{stages}$. The number of machines per stage varies from 1 to 3 machine(s). According to their characteristics, these benchmark problems are classified into 13 groups. The notation of each instance is represented by three letters corresponding respectively: (*j*) to job, (*s*) to stage and (*a*, *b*, *c* or *d*) to the machine configuration of each stage. For example, a 10-jobs, 5-stages with the machine structure *d* is denoted "***j10s5d2***" (see Table 2). The difference between a machine structure *d1* and *d2*, for example, are the processing times. These latter are integer generated from the uniform distribution $[3, 20]$. The machine configurations are divided into four types as follows:

- structure *a* is composed of one machine at the middle stage and three machines at the other stages
- structure *b* is composed of one machine at the first stage and three machines at the other stages
- structure *c* is composed of two machines at the middle stage and three machines at the other stages
- structure *d* is composed of three machines at all the stages.

All the *a*, *b* and ***j10s10c*** structures are identified as easy problems. The rest of the problems are considered as hard problems (Carlier and Néron, 2000). The authors calculated the *LB* of all these problems and have limited the CPU computation time to 1600 seconds.

We tested 9 versions of *ACS-HFS* corresponding, on the one hand, to the 8 different heuristic values that differ in the dispatching rule they used (*SPT*, *LPT*, *LWKR*, *MWKR*, *SRT*, *LRT*, *EST*, *EFT*) and, on the other hand, to a version that does not use any heuristic value at all. Then for each algorithm version, five test runs are realized to each instance problem and the best $C_{\max}$ is kept. According to the obtained results (all the results are not reported in this paper), we can conclude that the heuristic value has an important impact on the solution quality. And, no heuristic value is systematically better than an other according to these instances, but some of them like the heuristic values *SPT*, *MWKR*, *SRT* provide better results for some hard instance problems than the others.

Néron *et al.* (2000) solved the above benchmark problems by a branch and bound procedure (*B&B*), and Alaykýran *et al.* (2007) solved only 63 of them by an improved ant system (AS). The solutions found by our best *ACS-HFS* version algorithm (see Table 2) are compared to these result methods in order to evaluate the performance of our approach.

Problem	***LB of C_{max}***	***ACS-HFS***	***Improved AS***	***B&B***	***%Deviation***		
		C_{max}	***C_{max}***	***C_{max}***	***ACS-HFS***	***Improved AS***	***B&B***
j10s5a2	88	88	88	88	0	0	0
j10s5a3	117	117	117	117	0	0	0
j10s5a4	121	121	121	121	0	0	0
j10s5a5	122	122	124	122	0	1,6	0
j10s5a6	110	110	110	110	0	0	0
j10s5b1	130	130	131	130	0	0,8	0
j10s5b2	107	107	107	107	0	0	0
j10s5b3	109	109	109	109	0	0	0
j10s5b4	122	122	124	122	0	1,6	0
j10s5b5	153	153	153	153	0	0	0
j10s5b6	115	115	115	115	0	0	0
j10s5c1	**68**	**68**	**68**	**68**	**0**	**0**	**0**
j10s5c2	**74**	**75**	**76**	**74**	**1,3**	**2,7**	**0**
j10s5c3	**71**	**72**	**72**	**71**	**1,4**	**1,4**	**0**
j10s5c4	**66**	**66**	**66**	**66**	**0**	**0**	**0**
j10s5c5	**78**	**78**	**78**	**78**	**0**	**0**	**0**
j10s5c6	**69**	**69**	**69**	**69**	**0**	**0**	**0**
j10s5d1	**66**	**66**	**#**	**66**	**0**	**#**	**0**
j10s5d2	**73**	**73**	**#**	**73**	**0**	**#**	**0**
j10s5d3	**64**	**64**	**#**	**64**	**0**	**#**	**0**
j10s5d4	**70**	**70**	**#**	**70**	**0**	**#**	**0**
j10s5d5	**66**	**66**	**#**	**66**	**0**	**#**	**0**

Table 2. Solutions of test problems(bold problems are hard problem)
(# refers to a none solved problem)

Problem	***LB of C_{max}***	***ACS-HFS***	***Improved AS***	***B&B***	***%Deviation***		
		C_{max}	***C_{max}***	***C_{max}***	***ACS-HFS***	***Improved AS***	***B&B***
j10s5d6	**62**	**62**	**#**	**62**	**0**	**#**	**0**
j10s10a1	139	139	#	139	0	#	0
j10s10a2	158	158	#	158	0	#	0
j10s10a3	148	148	#	148	0	#	0
j10s10a4	149	149	#	149	0	#	0
j10s10a5	148	148	#	148	0	#	0
j10s10a6	146	146	#	146	0	#	0
j10s10b1	163	163	163	163	0	0	0
j10s10b2	157	157	157	157	0	0	0
j10s10b3	169	169	169	169	0	0	0
j10s10b4	159	159	159	159	0	0	0
j10s10b5	165	165	165	165	0	0	0
j10s10b6	165	165	165	165	0	0	0
j10s10c1	113	116	118	127	2,6	4,4	12,4
j10s10c2	116	119	117	116	2,6	0,9	0
j10s10c3	98	119	108	133	21,4	10,2	35,7
j10s10c4	103	124	112	135	20,4	8,7	31,1
j10s10c5	121	129	126	145	6,6	4,1	19,8
j10s10c6	97	109	102	112	12,3	5,1	15,5
j15s5a1	178	178	178	178	0	0	0
j15s5a2	165	165	165	165	0	0	0
j15s5a3	130	130	132	130	0	1,5	0
j15s5a4	156	156	156	156	0	0	0
j15s5a5	164	164	166	164	0	1,2	0
j15s5a6	178	178	178	178	0	0	0
j15s5b1	170	170	170	170	0	0	0
j15s5b2	152	152	152	152	0	0	0
j15s5b3	157	157	157	157	0	0	0
j15s5b4	147	147	149	147	0	1,3	0
j15s5b5	166	166	166	166	0	0	0
j15s5b6	175	175	176	175	0	0,6	0
j15s5c1	**85**	**85**	**85**	**85**	**0**	**0**	**0**
j15s5c2	**90**	**91**	**90**	**90**	**1,1**	**0**	**0**
j15s5c3	**87**	**87**	**87**	**87**	**0**	**0**	**0**
j15s5c4	**89**	**92**	**89**	**89**	**3,4**	**0**	**1,1**
j15s5c5	**73**	**80**	**73**	**84**	**9,6**	**0**	**15,1**
j15s5c6	**91**	**91**	**91**	**91**	**0**	**0**	**0**
j15s5d1	**167**	**167**	**167**	**167**	**0**	**0**	**0**
j15s5d2	**82**	**86**	**86**	**85**	**4,9**	**4,9**	**3,7**
j15s5d3	**77**	**89**	**83**	**96**	**15,6**	**7,8**	**24,7**
j15s5d4	**61**	**90**	**84**	**101**	**47,5**	**37,7**	**65,6**
j15s5d5	**67**	**85**	**80**	**97**	**26,9**	**19,4**	**44,8**
j15s5d6	**79**	**87**	**79**	**87**	**10,1**	**0**	**10,1**
j15s10a1	236	236	236	236	0	0	0
j15s10a2	200	200	200	200	0	0	0
j15s10a3	198	198	198	198	0	0	0
j15s10a4	225	227	228	225	0,9	1,3	0
j15s10a5	182	182	182	183	0	0	0,5
j15s10a6	200	200	200	200	0	0	0
j15s10b1	222	222	222	222	0	0	0
j15s10b2	187	187	188	187	0	0,5	0
j15s10b3	222	222	224	222	0	0,9	0
j15s10b4	221	221	221	221	0	0	0
j15s10b5	200	200	#	200	0	#	0
j15s10b6	219	219	#	219	0	#	0
			%solved problem *(the 77 problems)*		77,92		81,81
			Average for all the 77 problems		2,45		3,637

Table 2. (Continued)

Table 2 summarised the best $C_{\max}$ values and the %Deviation found (for all the 77 instances) by both of our *ACS-HFS* algorithm and the *B&B* method (Carlier and Néron, 2000) and (for only 63 instances) by the improved *AS (*Alaykýran *et al.*, 2007). About these results, we can give the following observations:

- For easy problems, the proposed *ACS-HFS* algorithm found better results than for hard ones. Indeed, it solved 46 of the 53 easy problems and 14 of the 24 hard problems.
- For all the 77 benchmark problems, the average deviation of the *ACS-HFS* algorithm is 2,45 while the average deviation of the *B&B* is 3,637.

These results confirm that the machine configurations have an important effect on the complexity of problems which affect the solution quality.

In table 3, we note that according to the 63 benchmark problems considered by Alaykýran *et al.* (2007), the average deviation difference from the *LB* for our *ACS-HFS* is 2,99. Our *ACS-HFS* method provides better %deviation than *B&B* method (4,45) but the *B&B* reached the *LB* values for 81% of the considered problems, while our method reached the *LB* values for 73,01%. In comparison with the improved AS, results shown that the %deviation from the *LB* of this latter method is equal to 2,1 but the improved AS could reach the *LB* for only 65% of the considered sample examples.

Method	**%solved problem**	**%deviation**
ACS-HFS	73,01	2,99
Improved *AS*	65	2,1
B&B	81	4,45

Table 3. Performances of the methods.

In table 4, we note that for hard problems, the improved AS outperformed both of our ACS-HFS and the B&B methods. The difference between the %deviation of the improved AS and the two methods (respectively our ACS-HFS and the B&B methods) are 2,67% and 5,07%. We remark also that the average deviation difference from the LB for our ACS-HFS is better than the B&B one for both easy and hard problems.

Method	**Easy problems**		**Hard problems**	
	%solved	%deviation	%solved	%deviation
ACS-HFS	84,44	1,48	55,55	6,77
Improved AS	64,44	1,3	66,67	4,1
B&B	86,66	2,56	61,11	9,17

Table 4. Relative Efficiency of the methods.

We can also conclude that our *ACS-HFS* method which uses a local search is more successful than the same method without a local search.

***Processing time*:** We can say that our proposed approach *ACS-HFS* provides solutions with a reasonable time (less than 1 second for the majority of the easy problems). For hard problems, the smallest CPU time is obtained for the *j15s5d1* problem (0,031s). For the unsolved optimally problems, the CPU time is less than 1000s. We note that the CPU time depends on the number of the algorithm iterations and on the number of application of the local search method.

5.2. Randomly generated benchmarks

In this part, 11 randomly generated examples have been tested to evaluate the quality of our method when we increase the number of jobs n($n \geq 15$). This latter is chosen to be equal to 25, 50 and 100. The three- and five-stage HFS are considered. The machine configurations are chosen to be a combination of $m_s = 5,10$ ($s = 1,2,3$) (see Table 5) or to be randomly generated from $m_s \in [1,5]$ ($s = 1,2,3,4,5$) (see Table 6). The processing times are generated as uniformly distributed random numbers on the interval $[1,100]$.

For each instance problem, five test runs are realized. Our randomly generated examples were tested according to the experimental results of the literature benchmarks ones. For each series, we used only the heuristic values *SPT* and *MWKR* to calculate solutions.

To our knowledge, no optimal solutions for the HFS problem have been presented on literature for large-scale instances. Then, to compare our experimental results, we have to determine a *LB* for each instance to evaluate the effectiveness of the solution. In this experimental evaluation, we apply the *LB* proposed by Santos et al. (1995). Table 5 and 6 show the percentage deviations for different stage configurations and job numbers.

	%deviation		
m_1, m_2, m_3	$n = 25$	$n = 50$	$n = 100$
5,5,5	8,25	14,07	8,97
5,5,10	4,84	5,86	3,12
5,10,5	4,92	8,39	2,02
5,10,10	1,31	4,66	1,47
10,5,5	9,09	14,94	3,63
10,5,10	4,88	9,52	6,95
10,10,5	3,51	2,02	1,17
10,10,10	0	9,5	19,29
average	4,6	8,62	5,8

Table 5. The percentage deviation for the 3-stage HFS instances.

	%deviation		
m_1, m_2, m_3, m_4, m_5	$n = 25$	$n = 50$	$n = 100$
1, 3, 3, 2, 5	0	0	0,49
5, 3, 1, 2, 2	0,19	0	0
2, 3, 5, 2, 4	0,5	0	1,11
average	0,23	0	0,53

Table 6. The percentage deviation for the 5-stage HFS instances

According to the obtained results, we can conclude that:

- The instances with the largest machines numbers in all the stages are the ones producing the largest %deviation.
- In the average, the procedure deviate from the considered lower bound by no more 8,62%.
- Other lower bounds may be further used to test our proposed method. In fact, the effectiveness of the lower bounds of Santos *et al.*(1995) have been tested by the authors on small to medium sized problems. In addition, the global lower bounds turn to be less tight when the number of stages becomes larger (Jin *et al*., 2006).
- The machine configurations have an important effect on the complexity of problems which affect the solution quality.
- The %deviation depends not only on the machine configurations but also on the data of the problem (such as the processing times).
- As concluded in §5.1, the CPU time depends on the number of the algorithm iterations and on the number of application of the local search method. It increases by increasing the number of jobs *n* and the numbers of machines per stage.
- We can say that our results show the effectiveness of our procedure.

6. CONCLUSIONS

This paper presents an ant colony optimization meta-heuristic approach for solving the hybrid flow shop scheduling problem. The complexity of this problem is *NP*-hard. To solve it, we propose a method which consists on solving the problem by phases. The first phase is used to solve the assignment sub-problem via some static heuristics. The second phase is dedicated to solve the sequencing sub-problem by adapting the ACS algorithm. Adding a local search method to our algorithm enhances the performance of the algorithm.

Our *ACS-HFS* method have been tested on instances taken from the literature (Carlier and Néron, 2000) and on randomly generated instances. For the former instances, the percentage deviations from lower bounds are provided and the obtained results have been compared with *B&B* and the improved *AS*. The latter instances are used to verify the proposed method when we increase the number of jobs. Lower bounds proposed by Santos *et al.* (1995) have been used to determinate the percentage deviations.

Extensive computational experiments suggest that our *ACS-HFS* method yields promising and good results. Our approach shows also the impact of the heuristic values in the quality of the solution. Likewise the parameter settings are important factors. They play an important rule in solutions quality.

Further work needs to be focused on improving the efficiency of the *ASC-HFS* problem. The use of alternative pheromone models and heuristic values may be improving the accuracy of our *ACS-HFS* approach. The proposed method may be also used in hybrid with other meta-heuristics in order to achieve better results. It may also further important to use or propose other global lower bounds for the hybrid flow shop scheduling problem to test the effectiveness of our meta-heuristic.

REFERENCES

Alaykýran, K., O. Engin, and A. Döyen, 2007. Using ant colony optimization to solve the hybrid flow shop sceduling problems. *The International Journal of Advanced Manufacturing Technology*, London Springer.

Baker, K.R., 1974. Introduction to sequencing and scheduling. John Wiley and Sons, New York.

Blazewicz, J., K.H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, 1996. *Scheduling computer and manufacturing processes*, Berlin, Springer.

Blum, C. and M. Sampels, 2004. An Ant Colony Optimization Algorithm for Shop Scheduling Problems. *Journal of Mathematical Modelling and Algorithms*, 3, p. 285-308.

Blum, C. and M., Sampels, 2002. Ant Colony Optimization for FOP Shop Scheduling: a case study on different pheromone representations. *Proceeding of the conference on Evolutionary Computation*, 2, p. 1558-1563.

Carlier, J. and E. Néron, 2000. An exact method for solving the multi-processor flow-shop. *RAIRO-RO*, 34(1), p. 1-25.

Colorni, A., M. Dorigo, and V. Maniezzo, 1991. Distributed Optimization by Ant Colonies. *Proceedings of European Conference on Artificial Life*, Paris, France.

Colorni, A., M. Dorigo, V. Maniezzo, and M. Trubian, 1994. Ant System for Job-Shop Scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science (JORBEL)*, 34(1), p. 39-53.

Den Besten, M., T. Stützle, and M. Dorigo, 2000. Ant Colony Optimization for the Total Weighted Tardiness Problem. *In M. Schoenauer et al., editors, Proceedings of* the Sixth *International Conference on Parallel Problem solving from Nature(PPSN VI), volume 1917 of lecture notes on Computer Science*, Berlin, Springer Verlag, p. 611-620.

Dorigo, M. and L.M. Gambardella, 1997. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1), p.53-66.

Engin, O. and A. Döyen, 2004. A New approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Generation Computer Systems*, 20(6), p. 1083-1095.

Gagné, C., W.L. Price and M. Gravel, 2002. Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence dependent setup times. *Journal of the Operational Research Society*, 53(8), p. 895-906.

Gajpal, Y., C. Rajendran, and H. Ziegler, 2006. An ant colony algorithm for scheduling in flowshops with sequence-dependent setup times of jobs. *International Journal of Advanced Manufacturing Technology*, 30(5-6), p. 416-424.

Ghedjati, F. and J.C. Pomerol, 2000. Dynamical heuristic for the job-shop scheduling problem with parallel machines and precedence constraints. European Journal of automation, 34(5), p. 623-645.

Gourgand, M., N. Grangeon, and S. Norre, 2000. Metaheuristics for the deterministic hybrid flow shop problem. *RAIRO-APII-JESA*, 34, p. 1107-1135.

Graham, R.L., E.L. Lawler, and A.H.G. Rinnooy Kan, 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5, p. 287-326.

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

Jin, Z.H., K. Ohno, T. Ito, and S.E. Elmaghraby, 2002. Scheduling Hybrid Flowshops in Printed Circuit Board Assembly Lines. *Production and Operations Management*, 11(1), p. 216-230.

Jin, Z.H., Z. Yang, and T. Ito, 2006. Metaheuristic algorithms for multistage hybrid flowshop scheduling problem. *International Journal of Production Economics*, 100, p. 322-334.

Nait Tahar, D., F. Yalaoui, L. Amodeo, and C. Chu, 2005. An ant colony system minimizing total tardiness for hybrid job shop scheduling problem with sequence dependent setup times and release dates. *International Conference on Industrial Engineering and Systems Management*, Marrakech, Morocco. 10 p.

Negenman, E.G., 2001. Local search algorithms for the multiprocessor flow shop scheduling problem. European Journal of Operational Research, 128(1), p. 147-158.

Néron, E., Baptiste, P., and Gupta, J.N.P, 2001. Solving hybrid flow shop problem using energetic reasoning and global operations. *Omega*, 29, p. 501-511.

Nowicki, E. and C., Smutnicki, 1998. Scheduling in a two stage manufacturing process. *European Journal of Operational Research*, 106, p. 226-253.

Portmann, M-C., A. Vignier, D. Dardilhac, and D. Dezalay, 1998. Branch and bound crossed with GA to solve hybrid flowshops. *European Journal of Operational Research*, 107 (2), p. 389-400.

Sankar, S.S., S.G. Ponnambalam, V. Rathinavel, and M.S. Visveshvaren, 2005. Scheduling in parallel machine shop: an ant colony optimization approach. *Industrial Conference on Industrial Technology, ICIT*, p. 276-280.

Santos, D.L., J.L. Hunsucker, D.E. Deal, 1995. Global lower bounds for flow shops with multiple processors. *European Journal of Operational Research*, 8, p. 112-120.

Stützle, T., 1998. An ant approach to the flow shop problem. *In Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing (EUFIT'98)*, Aachen, Germany, p. 1560-1564.

Stützle, T. and H. Hoos, 1997. Improvement in the ant system: introducing min-max ant system. *In proceedings of the international conference on Artificial Neuronal Networks and Genetic algorithms*, Vienna, Springer, p. 266-274.

Vignier, A., J.C. Billaut, and C. Proust, 1999. Les Problèmes d'ordonnancement de type flow-shop hybride: état de l'art. RAIRO-RO, 33(2), p. 117-83.

Ying, K-C. and C-J. Liao, 2004. An ant colony system for permutation flowshop sequencing. *Computers & Operations Research*, 31(5), p. 791-801.

Ying, K-C. and S-W. Lin, 2006. Multi-heuristic desirability ant colony system heuristic for non permutation flowshops scheduling problems. *International Journal of Advanced Manufacturing Technology*.

Zhang, J., X. Hu, X. Tan, J.H. Zhong, and Q. Huang, 2006. Implementation of an Ant Colony Optimization technique for job shop scheduling problem. *Transactions of the Institute of Measurement and Control*, 28, p. 93-108.