FLEXIBLE JOB SHOP SCHEDULING WITH SEQUENCE-DEPENDENT SETUP AND TRANSPORTATION TIMES BY ANT COLONY WITH REINFORCED PHEROMONE RELATIONSHIPS

AUTHOR: ANDREA ROSSI[1]

[1] *Department of Civil and Industrial Engineering, Università di Pisa, Via Bonanno Pisano, 25/B, Pisa, 56126, ITALY*

- Corresponding author: Tel. ++39 50 2218123, Fax ++39 502218140; E-mail: andrea.rossi@dimnp.unipi.it

HIGHLIGHTS:

- ACO enhances machine assigning/sequencing constraints and dynamic visibility function
- Statistical tests suggest the superiority of the ACO with systems in literature
- ACO shows no significant difference to schedule sequence-dependent/independent setups
- Tests suggest the system is no significantly influenced by low setup and feeding times
- Experiments suggest that the CPU time slight increases with problem complexity

Abstract

This paper proposes a swarm intelligence approach based on a disjunctive graph model in order to schedule a manufacturing system with resource flexibility and separable setup times. Resource flexibility assigns each operation to one of the alternative resources (assigning sub-problem) and, consequently, arranges the operation in the right sequence of the assigned resource (sequencing sub-problem) in order to minimize the makespan. Resource flexibility is mandatory for rescheduling a manufacturing system after unforeseen events which modify resource availability. The proposed method considers parallel (related) machines and enforces in a single step both the assigning and sequencing sub-problems. A neighboring function on the disjunctive graph is enhanced by means of a reinforced relation-learning model of pheromone involving more effective machine-sequence constraints and a dynamic visibility function. It also considers the overlap between the jobs feeding and the machine (anticipatory) setup times. It involves separable sequence-independent and dependent setup phases. The algorithm performance is evaluated by modifying the well-known benchmark problems for job shop scheduling. Comparison with other systems and lower bounds of benchmark problems has been performed. Statistical tests highlight how the approach is very promising. The performance achieved when the system addresses the complete problem is quite close to that obtained in the case of the classical job-shop problem. This fact makes the system effective in coping with the exponential complexity especially for sequence dependent setup times

Keywords: *parallel machines, metaheuristics, swarm systems, benchmark problems, computation time*

1. Introduction

A common aim in the practical job shop environment is to improve resource flexibility and setup lag times. Resource flexibility deals with flexible (or hybrid) job shop scheduling (FJS) where alternative resources are present to increase performance, to manage preventive maintenance or to tackle breakdown and other unforeseen events which modifies resource availability. The FJS problem is thus to determine both an assignment of each operation to one of the alternative resources (*assignment sub-problem*) and an ordering of the operations on each assigned resource (*sequencing sub-problem*) with the aim of optimising an objective function.

The FJS problem arises in at least two types of workshop. The first is a flexible manufacturing system where a small number of multi-purpose machines are equipped with different tools and a number of modes (*multiple modes*) are allowed to perform each operation. A FJS has a *total flexibility* if any operation can be processed by each machine present in the system. This method gives the required routing flexibility but the managing of machine capabilities to meet the tolerances of design (process planning) is a very difficult problem.

In the great majority of practical industrial applications, however, resource flexibility is combined with scheduling operations on alternative (identical) machines. It consists of workshops with a partition of available machines into groups of parallel machines tools. The machines of the same group (e.g. lathes, milling machines, washing/sterilization machines, measuring machines, assembly robots, etc.) as *related*: they group the manufacturing capability in order to process a set of technologically similar operations and hence, for example, they include equal processing and setup times (Stecke and Raman, 1995). If the jobs have an identical routing among the groups, the problem is the hybrid (non-permutation) flow-shop scheduling.

Job shop scheduling with the objective of makespan minimization $J_m||C_{\max}$, which only deals with the sequencing problem, is strongly NP-hard (Garey et al, 1976). An extensive and rapidly growing series of approaches are proposed; nevertheless, only a few special cases can be optimally solved with effective computing times (see Jain and Meeran, 1999; Blazewicz et al., 1996 for a review). As it is an extension of job shop scheduling, the flexible job shop scheduling $FJ_m||C_{\max}$ is NP-hard as well. Flexible job shop scheduling has only been treated in recent literature by a number of approaches where the routing flexibility is achieved by multiple modes with partial flexibility (Brucker and Schlie, 1990; Hurink et al., 1994; Brucker and Thiele, 1996; Dauzère-Pérès et al., 1998; Mastrolilli and Gambardella, 2000; Kacem et al. 2002; Kumar et al. 2003; Chan et al. 2006). In addition, Kacem et al. (2002) and Chan et al. consider FJS with total flexibility.

Compared with the extensive research on FJS where the routing flexibility is achieved by multiple modes, in the last years the systems and the applications to solve the parallel machine job shop scheduling problem has not received sufficient attention. Besides, the majority of FJS systems assume release date of jobs and resources, operation setup and job transportation (travel) times as negligible or part of the processing time. While these assumptions simplify the analysis in certain applications, they adversely affect the solution quality for many applications which require explicit treatment of these time lags. Such applications have motivated an increasing interest to include setup considerations, in order to reduce costs.

According to the $\alpha|\beta|\gamma$ notation of Graham et al. (1979), the problem under consideration can be denoted by $PJ_m(k)|\ s_{jk}\ ,prec\ |C_{max}$, where the field α denotes a job shop with k parallel resource per group and m groups, the field β indicate the presence of sequence-dependent setup times and linear routings, i.e. the occurrence of simple precedence constraints in the job routing and, the field γ denotes the makespan as the adopted measure of performance. In such a system, an operation is subjected to the following lag times: i) sequence-dependent (*SD*) setup, the setup which depends on the previous operation processed on the resource; ii) sequence-independent (*SI*) setup, the setup which depends on the previous operation in the job routing (i.e job transportation) (Allahverdi et al., 2008). *Overlapping* among transportation and processing times and *anticipatory setup*, which causes the part not to be necessarily available on the resource during the setup period, involves separable *SI* and *SD* setup phases.

A number of job shop scheduling approaches assume the material handling system as a further resource where travelling operations involve non-negligible transportation times. Thus, the material handling system can be scheduled together with the resources, in order to avoid transportation costs which could influence the makespan (Artigues and Roubellat, 2001; Hurink and Knust, 2005). In such approaches, the scheduling algorithm must schedule twice the number of operations and one (or more) further resource included in the material handling system. Among the minor considerations, resource setup and transportation times can be seen as related to a single *operation setup phase* which includes separable *SD* and *SI* times. This approach reduces the number of operations in the system because no additional resource is used to model transportation times. Ivens and Lambrecht (1996) consider separable sequence-independent setup and travel times in the case of multi-stage multiprocessor flow-shop scheduling and non-linear routing. They extend the disjunctive graph (*digraph*) representation for job shop scheduling, originally proposed by Roy and Sussman (1964). Blazewicz et al. (1996) state that the digraph model is becoming the standard model for scheduling applications because it is more efficient than Gantt diagrams to describe knowledge for optimisation search techniques. Rossi and Dini (2001) propose a $PJ_m(k)|\ prec\ |C_{max}$ where separable setup and transportation times are related to a single operation setup phase and the problem knowledge for an evolutionary approach is still modelled by a Gantt diagram. A digraph approach to a case of study of the parallel machine job shop scheduling with setup lag times is proposed by Rossi and Dini (2007).

Artificial life methods have been developed in order to tackle the computational complexity of hard problems by means of a sort of implicit parallelism which offers a population-based iterative algorithm. This offers the possibility of obtaining a reactive, robust algorithm, which is basic for an industrial dynamic production process (De Jong and Spears, 1995). The Ant Colony Optimization (ACO: Bonebeau et al., 2000) is a promising metaheuristic and an emerging class of research, dealing with swarm intelligence, a set of artificial life methods which exploit the experience of an ant colony as a model of self-organisation in co-operative food retrieval by means of a proper pheromone trail model. The pheromone trail is the basic mechanism of communication among real ants and it is mimicked by the ACO in order to find the shortest path connecting source and destination on a weighted graph which represents the optimization problem. As soon as a path is generated, the artificial ant deposits on the arc a further amount of pheromone proportional to the path length and a pheromone decay routine is performed to prevent stagnation.

$J_m||C_{\max}$ has been approached by an ant system (AS) proposed by Colorni et al. (1994). Kumar et al. (2003) propose an AS to approach the $FJ_m|\ s_{jk}\ ,prec|\ C_{max}$ problem. Nevertheless, ant systems has been improved by Ant Colony Optimization (ACO). Two main classes of ACO systems are proposed in literature in order to improve intensification and diversification mechanisms of ant systems: the Ant Colony System (ACS: Dorigo and Gambardella, 1997) and the MinMax Ant System (MMAS: Stutzle and Hoos, 2000). A MinMax Ant System was proposed by Blum and Sampels (2004) for solving a kind of job shop scheduling in which some routing constraints are removed. This ACO hybridizes some components of the current state-of-the-art system for job shop scheduling, the tabu search proposed by Nowicki and Smutnicki (1996), in order to outperform the *pure*-MMAS. Besides, no ACO has been extended to approach resource flexibility.

This paper describes an ACO approach to $PJ_m(k)| s_j = s_{jk}, prec |C_{max}$ problem, where the reconfiguration tasks of the resources of the same group are standardized with a predetermined number of procedures. This standardization is very important in real manufacturing systems for the efficient planning of the clamping or the batching of the parts to be produced. It is based on the digraph model of the flexible job shop scheduling problem with separable transportation and sequence-dependent setup times. The proposed system uses an algorithm similar to the list scheduler (originally proposed by Giffler and Thompson, 1960, for classic $J_m||C_{\max}$) to generate a feasible schedule on the digraph by visiting every operation once and only once. Here, the aim is to minimize the makespan, although an amount of results is independent of the selected objective function.

2. Job shop Scheduling with Resource Flexibility and Separable Setup Times

In FJS, n jobs have to be scheduled on m resources in accordance with its linear routing represented by a sequence of $l_i \leq m$ operations, $O_{i\,j\,r}$; each of these has to be processed as the r^{th} operation on a single resource selected within a set of resources $M_{i\,j}$ ($| M_{i\,j} | \leq m$), with a setup activity $f_{i\,j}$, which takes the time $t(f_{i\,j})$, and a processing time $t_{i\,j\,r}$; $st(O_{i\,j\,r})$ and $t(O_{i\,j\,r})$ denote respectively the starting and the completion time of the operation.

No resource can process more than one operation at a time; no operation $O_{i\,j\,r}$ can start until $O_{i\,j\,r-1}$ is completed or can stop after it starts; finally, an operation must be processed by one, and only one, resource.

In order to process the entire set of planned operations, the system includes F dedicated setup activities, grouped per resources, $F_1,..,F_m$. A resource j includes all the equipment capable of performing the setup activity $f_{i\,j}$ of the set F_j. Each job i and each resource j are subjected to, respectively, a release date, D_i and a deadline d_j. Finally, a material handling system is able to offer the required flexibility in order to move a part through the system. A transportation time matrix δ includes the times to move a job among the stations.

The quantity $k = \min_{i\,j} |M_{ij}|$ represents the degree of *parallelization* capability of the system. Job shop scheduling is a particular case of FJS where k=1. The assignment of operations to a resource of a pool gives a sort of further flexibility, and hence an increase of complexity, in addition to the flexibility represented by the possibility of sequencing operations on the resources. In particular, the FJS has a total flexibility if $k = m$ and has a partial flexibility if $k<m$.

The $PJ_m(k_1,.., k_m)||C_{\max}$, where k_j is the number of resources of group j, is a restriction of the FJS in which the sets of related resources $M_{i\,j}$ is a *total grouping*:

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\left|M_{i\,j}\right| = \sum_{i=1}^{m} k_j = m \tag{1}$$

$$\left(\bigcap_{i=1}^{n} M_{i\,w}\right)\bigcap\left(\bigcap_{i=1}^{n} M_{i\,z}\right) = \varnothing, \quad w, z = 1,..,m,\ w \neq z \tag{2}$$

The degree of parallelization capability is represented by the quantity $k = \min_{j=1,..,m} k_j$. The $PJ_m(k)||C_{\max}$ has a degree k of parallelization; as a consequence, $PJ_m(1,.., 1)||C_{\max}$ is the classic job shop scheduling, $J_m||C_{\max}$, and reflects the fact that it has non-parallelization capability.

The starting time of an operation is evaluated by considering: i) anticipatory (sequence-dependent) setup and overlapping of processing and transportation times; ii) the semi-active schedule, where operations are processed as early as possible. This necessarily means that the starting time of an operation depends on the longest time between the *SI* and *SD* setup. Hence, setting $O_{i'\,j\,r'}$ the previous operation in the queue of resource h of pool j, the starting time of the current operation is evaluated by the following expression:

$$st(O_{i\,j\,r}) = \max\{t(SD), t(SI)\} = \max\{t(O_{i'j\,r'}) + t\,(f_{ij}),\ t_A + \delta(h, h_1)\} \tag{3}$$

where t_A is the instant of availability of the material handling system; $t(f_{ij})$ and $\delta(h, h_1)$ are the *lag times* for, respectively, *SD* and *SI* setup. The first setup on every resource starts from the resource release date (i.e. $t(O_{i'j\,r'}) = d_h$). By expression (1), *SI* setup depends on the availability and the time of the material handling system (i.e. $t_A \geq t(O_{i\,j\,r-1})$ if r>1, $t_A \geq D_j$, otherwise).

3 Disjunctive graph representation

Disjunctive graph representation is becoming the standard model for job shop scheduling; moreover, it offers a scheme for sequence-dependent setup times (Brucker and Thiele, 1996) and FJS (Dauzère-Pérès and Paulli, 1997). A disjunctive graph is represented by:

$$DG = (N, A, E_{\mathrm{j}}) \tag{4}$$

where N is the set of operations plus the dummy start and finishing operations 0 and *; A is the set of conjunctive arcs between every pair of operations on a job routing, between 0 and every first operation on a routing, and between every last operation on a routing and *; E_{j} is the set of disjunctive arcs between pairs of operations, $O_{*\mathrm{j}*}$, that have to be processed on the same resource j (j=1,..,m); it also includes disjunctive arcs between 0 and $O_{*\mathrm{j}*}$, between $O_{*\mathrm{j}*}$ and * and between 0 and * for all E_{j}. Figure 1 shows an example of a FJS with parallel resources (Figure 1.a). In the former, each operation O_{ij*} is connected with disjunctive arcs belonging to each alternative resource of the set M_{ij}. In the latter case, each arc which connects a pair of operations $O_{*\mathrm{j}*}$ to be processed on the same pool j, is replicated k_j times (as many times as the number of resources of pool j). The disjunctive graph is modified by including the sets $E_{\mathrm{j\,h}}$, i.e. $DG = (N, A, E_{\mathrm{j\,h}})$; $E_{\mathrm{j\,h}}$ is the set of disjunctive arcs between pairs of operations, $O_{*\mathrm{j}*}$, that have to be processed on the same resource h (h=1,..,k_j) of the same pool j (j=1,..,m). Thus all and only the k_j disjunctive arcs of $E_{\mathrm{j\,h}}$, are connected to every operation $O_{*\mathrm{j}*}$. In general, a $PJ_m(k)||C_{\max}$ problem is represented by m pools of k kinds of arcs E_{jh}; as the arcs of E_{jh} connected with n nodes are $n(n\text{-}1)$, the number of disjunctive arcs in the digraph is $O(kmn^2)$. As a consequence in the FJS with total flexibility, the number of disjunctive arcs is $O((mn)^2)$.

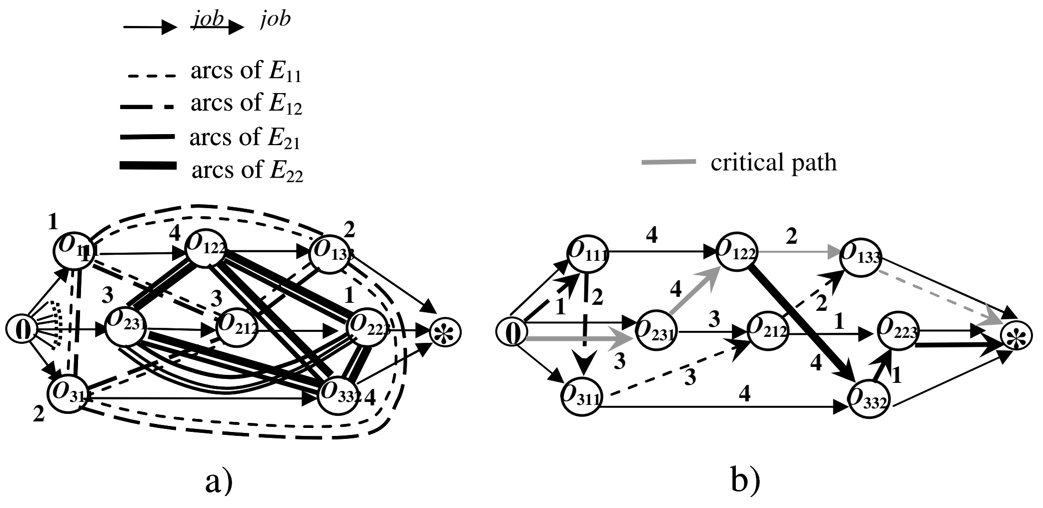

Figure 1 – Example of FJS with replicates resources: a) disjunctive graph; b) acyclic conjunctive graph; in particular, a $PJ_m(k)||C_{\max}$ with n=3 and m=2 is showed.

A finite sequence of conjunctive (directed) arcs between two operations is called a *path*. The *length* of an arc is equal to the processing time of the operation at which it ends. The *path length* is equal to the sum of the lengths of its arcs. A path which starts from 0 and ends at * is the *loading sequence* on a resource. A *cycle* is a path which starts and ends at the same operation. If no cycle is present in a conjunctive graph achieved by directing some disjunctive arcs and including all the operations, the conjunctive graph is *acyclic*, and the related loading sequences on the resources are a feasible schedule. In an acyclic conjunctive graph, the makespan of the feasible schedule *S* is the length of the critical path, i.e. the longest path between the dummy start and finishing operations, and it is indicated as *makespan*(*S*). Finally, the length of the longest path between the dummy start operation 0 and a given operation is its completion time. Figure 1.b shows an acyclic conjunctive graph and a related critical path achieved by directing arcs of the *DG*. The weighted path is achieved by moving the weights from an operation to the two arcs which end at it.

3.1 Integrating separable setup times

The digraph used to approach the problem with a given number of lag times, includes different kinds of nodes and further weights on the operations. This graph is represented by:

$$WDG=(N_F, A, E_{\mathrm{h\,j}}, W_{\mathrm{N}}) \qquad (5)$$

where the new component W_N is the weight on the nodes, represented by the four-dimensional array $W_N(O_{i\,jr})=(t_{i\,j},\ t\,(f_{ij}),\ \delta(h, h_1),\ D_i)$, which includes, respectively, processing time, lag times for *SI* and *SD* and release date of the related job; N_F is the partitioning of the set of nodes into the subsets of setup activities, where the operations which have the same setup are included in the same subset (i.e. they are associated to the same kind of node as proposed by Brucker and Thiele, 1996).

Figure 2 shows these aspects on the digraph of the example in Figure 1.a. The first and the second pools of resources perform two setup activities: $F_1 = \{f_{11},\ f_{21}=f_{31}\}, F_2 = \{f_{12}, f_{22}=f_{32}\}$; they are represented by two kinds of nodes. The third pool has one setup activity for each job; this is represented by three kinds of nodes. The release date of the resources is represented by the further weight on the arcs connected to the dummy operation 0; the release date of the jobs is represented by the further weight on the first operation on the routing. For each operation the two setup activities *SI* and *SD* are also showed in this example.

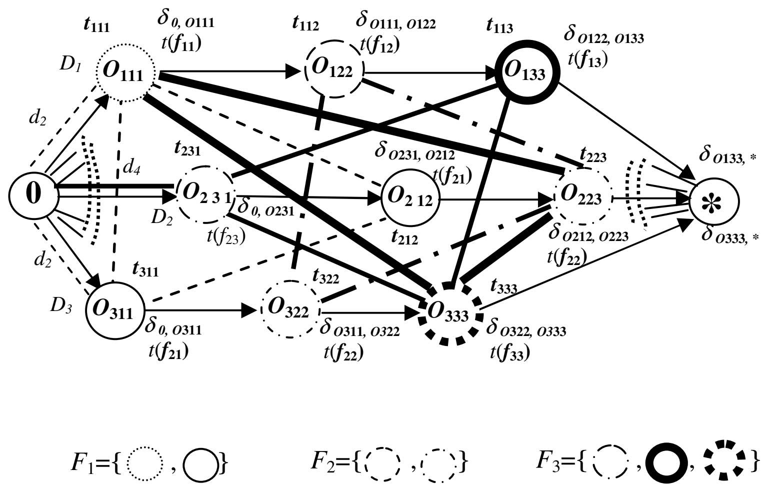

Figure 2 – Representation of flexible job-shop scheduling with separable transportation and sequence-dependent setup times. The operations which have the same setup activity are represented by the same kind of node.

3.2 Finding a feasible schedule

The proposed system uses a list scheduler (LS) algorithm to generate a feasible schedule on the digraph by visiting every operation once and only once. At every step, a node is connected to the acyclic conjunctive graph which represents the partial schedule, by means of a feasible move. A feasible move is a disjunctive arc which can be directed in the partial conjunctive graph without creating a cycle. The following is a pseudo-code description of the proposed LS algorithm. It possesses the required skills to generate a feasible schedule S with completion times placed on the related nodes and weights on the conjunctive arcs in accordance with expression (3).

LS algorithm for FJS with separate setup times

```
Input: a weighted digraph $WDG=(N_F, A, E_{hj}, W_N)$
$O \leftarrow \{O_{ijr} \mid i=1,..,n, j=1,..,m, r=1,..,m\}$ // $m$ operations for each job are considered
for each $w=1$ to $\Sigma_{i=1,...,n}\, l_i$ do
    1. Inizialization of Candidate Nodes: build the allowed list $AL_w$ for the current step $w$: $AL_w \leftarrow \{O_{ijr} \in O \mid O_{ijr-1} \cap O = \varnothing\}$
    2. Restriction: restriction of the allowed list by means of optimality criteria (i.e. active or non-delay schedule); let the candidate list $CL_w$ be the restricted allowed list.
    3. Inizialization of Feasible Moves: mark as a feasible move each disjunctive arc $(O_{i'jr'}, O_{ijr})$ of $E_j$ where $O_{ijr} \in CL_w$ and $O_{i'jr'}$ is the last operation of the loading sequence of resource $j$ (it creates the possibility for the candidate operation to become the new last operation of that loading sequence);
    4. Move Selection: select a feasible move $(O_{i'jr'}, O_{ijr})$ of $E_j$ by directing the related disjunctive arc ($O_{i'jr'}$ =‘dummy 0’,if $r=1$);
    5. Arcs Removal: remove all the remaining disjunctive arcs connected to $O_{i'jr'}$ (i.e. no other operation can be immediately subsequent to $O_{i'jr'}$ in the loading sequence); remove all the remaining disjunctive arcs of $E_h$ connected to $O_{ijr}$, i.e. $h \in M_{ij}$ and $h \neq j$ (i.e. no other loading sequence can include the operation);
    6. Computing length: the length of the arc $(O_{i'jr'}, O_{ijr})$ is evaluated as the sum of processing and lag times of node $O_{ijr}$ by means of expression (1) where $t_A = t(O_{ijr-1})$;
```

```
        7.    Transferring length: this length is placed on the related arc and on the arc of the job routing
              (arc of A) which ends at $O_{ijr}$; also, the completion time $t(O_{ijr}) = st(O_{ijr}) + t_{ij} = \max\{t(SI), t(SD)\} + t_{ij}$ is placed as a mark of the node $O_{ijr}$;
        8.    Updating Structures: update $O$ by removing operation $O_{ijr}$ : $O \leftarrow O \setminus O_{ijr}$;
end for
9.    Directing the remaining disjunctive arcs: the arcs are connected to the dummy operation *;
```

Output: the schedule *S* (i.e. *CG* with the completion times of the operations)

The LS algorithm generates in $O(mn(m+n))$ a *complete selection* of arcs of *WDG i.e. is* an acyclic conjunctive graph which includes all the nodes. This property results from the following considerations:

a) the achieved graph includes all the nodes: the main loop is performed |O| times and initially the candidate list includes all the nodes; at each iteration one and only one node is removed from candidate list (step 8);

b) the achieved graph is conjunctive: for each iteration, the selected feasible move is a conjunctive arc which ends at the node removed from candidate list (step 4); all the disjunctive arcs which starts from the first node of the conjunctive arc are removed (step 5);

c) the conjunctive graph is acyclic: each feasible move ends to a node which is in the candidate list, i. e. is a not scheduled operation (steps 1 and 3).

In order to evaluate the computational complexity of the LS algorithm, it can be noted that the main loop is performed $n{\cdot}m$ times and the most time-consuming step is the Arcs Removal step. The selection of a feasible move $(O_{i'jr'}, O_{ijr})$ entails that the following alternative arcs are removed:

i) *alternative sequencing* arcs: all the disjunctive arcs connected to the last operation $O_{i'jr'}$ in the loading sequence of resource *j*; they are at most *n*-1, one for each alternative job in order to approach the sequencing problem;

ii) *alternative assigning* arcs: all the disjunctive arcs of E_h , $h \in M_{ij}$ and $h \neq j$, connected to the candidate operation $O_{i\,jr}$; they are at most m-1, one for each alternative resource in order to approach the assigning problem;

As a consequence of these computational complexity considerations, the LS algorithm finds a feasible schedule by means of an implicit visit of a large number of disjunctive arcs. Another consequence is that the two sequencing and assigning decisional points are considered at the same time in the selection of a feasible move because, at the same time, it is both an alternative sequencing arc and alternative assigning arc. Dauzère-Pérès and Paulli (1997) state that the neighbouring function that considers at the same time alternative sequencing arcs and alternative assigning arcs is more effective compared to the approaches where assigning and sequencing problems are considered separately.

Finally, the completion time of an operation evaluated by expression (3) is the length of the longest path which starts from the dummy node 0. In fact, a selected operation $O_{i\,j\,r}$ has as its predecessors the previous operation in the routing, $O_{i\,j,r-1}$, and the last operation in the loading sequence of resource j, $O_{i'j\,r'}$. From expression (3), the starting time of the operation is evaluated for the maximum time between $t(SD) = t(O_{i'j\,r'})+t(f_{ij})$ and $t(SI) = t_A + \delta(h, h_1)$. Two cases have to be considered: i) $t(SD) \geq t(SI)$; ii) $t(SD) < t(SI)$. In the first case the lag time is the sequence-dependent setup time $t(f_{ij})$; in the computing length step, this lag time is added to the processing time and this length is placed on the directed arc in the transferring length step. As a consequence, the longest path includes the directed arc and the predecessor node $O_{i'j\,r'}$. On the basis of the same considerations, in the latter case the longest path includes the arc of the job routing and the predecessor node $O_{i\,j\,r-1}$.

3.3 Pheromone trail model

The pheromone trail models for job shop scheduling applications are represented by a graph where the node represents the operations and the arc represents the possibility for the two nodes to be visited in some precedence order by the ant; examples of precedence order are: i) operations to be processed in sequence on a resource; ii) operations which have sub-sequential finishing times (i.e. no other operations are completed in the time interval which ranges between the first and the last operations considered). In particular, an emerging model of the pheromone trail, the *relation-learning* model, uses the precedence order i) and hence can be represented by the disjunctive graph *DG* (Blum and Sampels, 2004). In this representation, only the operations which have to be processed on the same resource are connected; hence it tackles the complexity which arises in previous models of pheromone (Colorni et al., 1994; Kumar et al., 2003), in which every operation is connected to the others. In the relation-learning model, an amount of pheromone is deposited on the arcs of *DG*. In particular, two values of pheromone amount, related to the two possible directions in which the ant proceeds, are associated with a disjunctive arc of *DG*. As a consequence, in the relation-learning model, each disjunctive arc $(O_{i'j r'}, O_{i j r})$ of E_j supports: i) an amount of pheromone $\tau(O_{i'j r'}, O_{i jr})$ which represents the desirability of including the feasible move $(O_{i'j r'}, O_{i jr})$ in the ant path (i.e. the desirability of assigning the loading sub-sequence $O_{i'j r'}$, $O_{i j r}$ to resource *j*); ii) an amount of pheromone $\tau(O_{i j r}, O_{i'j r'})$ which represents the desirability of including the feasible move $(O_{i j r}, O_{i'j r'})$ in the ant path.

3.4 Relation-learning Ant Colony Systems

In the relation-learning model, an ant visits the disjunctive graph by means of the list scheduler algorithm, producing a path (*path generation*) which starts from dummy operation 0 and ends at dummy operation * for each set E_j. The *ant path* is represented by the acyclic conjunctive graph *CG*. The length of a critical path on *CG* produced by ant *a* is the *makespan*(S_a), which represents an index of desirability of the schedule S_a.
A relation-learning ant colony system is an iterative population-based system where at each epoch. A colony of *ps* ants builds a step-by-step feasible schedule by selecting a feasible move with the transition probability rule which depends on the intensification and the diversification mechanisms of pheromone amount. The best ant so far S_b deposits on the ant path an amount of pheromone which is a function of the desirability of the schedule S_b (off-line pheromone rule). In this way the local search is strengthened with the colony stigmergy which enhances the effective move selection.
The process ends when an optimality condition is verified represented by the reaching of:

a) the optimal solution or

b) a number of epochs without improvement of the best solution (*stability condition*).

The following procedures allow path generation and off-line update rule:

i) *path generation* - at each construction step w, an ant selects the next feasible move, Z, from the set of feasible moves, by means of the following transition probability rule which is a function of both the visibility function, η, and the amount of pheromone τ on the related arc:

$$Z = \begin{cases} \arg\max_{O_{ijr} \in CL} \left\{ \tau(O_{ijr}, O_{i'jr'}) \cdot \eta(O_{ijr}, O_{i'jr'})^{\beta} \right\} & \text{if } q \leq q_0 \\ J, & \text{if } q > q_0 \end{cases} \tag{6}$$

where q is a random number which ranges in [0,1] and q_0 is the cutting exploration parameter ($0 \leq q_0 \leq 1$). If q is higher than q_0, the feasible move J is selected in accordance with the random proportional rule of AS:

$$P_a(J) = \frac{\tau(J)^{\alpha} \cdot \eta(J)^{\beta}}{\sum_{O_{ijr} \in CL_w} \tau(O_{ijr}, O_{i'jr'})^{\alpha} \cdot \eta(O_{ijr}, O_{i'jr'})^{\beta}}, \quad \text{where} \quad \alpha = 1 \tag{7}$$

After selecting feasible move (O_{ijr}, $O_{i'jr'}$), the ant applies the local updating rule which imposes laying on the arc the following negative amount of pheromone:

$$(O_{ijr}, O_{i'jr'}) = (1-\rho) \cdot \tau(O_{ijr}, O_{i'jr'}) + \rho \cdot \tau_0 \tag{8}$$

where the parameter ρ is the evaporation coefficient, $0 \leq \rho \leq 1$, and τ_0 is a small positive constant which is initially deposited on all the arc of the digraph. When an ant path is generated, the solution is taken to its local optimum by means of a neighbouring structure implemented by a steepest descent local search routine.

ii) *off-line pheromone update* - the best ant S_b lays the following amount of pheromone at the end of each epoch:

$$\tau(O_{ijr}, O_{i'jr'}) = (1-\rho) \cdot \tau(O_{ijr}, O_{i'jr'}) + \rho \cdot \mathit{makespan}(S_b)^{-1}, M \in S_b \tag{9}$$

The transition probability rule allows an *intensification* mechanism in order to select a node in the vicinity of the current best path denoted by the complete selection S*. If makespan (S_b)<makespan (S^*), the current best path is updated by the complete selection S_b.

The role of cutting exploration is to find a compromise between the random proportional rule (7) and a mechanism of exploring near the best path so far. By tuning parameter q_0 near 1, cutting exploration allows the activity of the system to concentrate on the best solutions (*exploitation activity*), whereas, when q_0 is close to 0, all the solutions are examined in probability (*exploration activity*). The pheromone updating rules (8) and (9) are achieved by means of convex combinations between the point $\tau(O_{i\,j\,r}, O_{i'j\,r'})$ and, respectively, the points τ_0 and $makespan(S_b)^{-1}$. Thus, the local updating rule involves a negative amount of pheromone deposited on the ant path, whilst the off-line updating rule entails that the closer the ant path is to the optimum path, the more positive the amount of pheromone which is laid. The first rule makes possible a *diversification* mechanism in order to produce promising alternative paths by the other ants of the colony, whilst the latter rule makes possible an intensification of the search in the vicinity of the best path.

4. The proposed approach

The proposed ant colony system is a Reinforced Relation-learning ACS (*RR-ACS*) where the relation-learning model of pheromone is modified in order to consider a new component: the positioning constraints of a feasible move within the loading sequence of the assigned resource (described in section 4.1).

RR-ACS also adopted: i) the list scheduler algorithm for path generation, ii) the local search with the neighbourhood structure proposed by Nowicki and Smutnicki (1996) and iii) the adaptive parameter of cutting exploration q_0 proposed by Kumar et. al. (2003).

The Selection Move step is performed by means of the transition probability function (6) while the Local Updating rule of the pheromone makes possible an effective stigmergy with the other ants of the colony, which, obviously, is not obtained by the LS algorithm. Finally, a novel method for both generating the candidate list and achieving a more profitable visibility function is described. The following algorithm implements the proposed system.

```
RR-ACS for flexible job shop scheduling with separable setup times
Input: a weighted digraph $WDG=(N_F, A, E_{h\,j}, W_N, W_E)$
// Initialization
for each edge of WDG, deposit a small constant amount of pheromone $\tau_0$
for each ant a, a=1 to ps, place the ant on a randomly chosen operation $O_{i\,j\,0}$
epoch ← 1; best_so_far, best_epoch ← MAXINT
// Main Loop
while "optimality condition is not satisfied" do
    // Epoch Loop
    for each ant a, a=1 to ps do
        // Path Generation
        $S_a \leftarrow \emptyset$;
        $O \leftarrow \{O_{i\,j\,r} \mid i=1,..,n, j=1,..,m, r=1,..,l_i\}$;
        for each $w=1$ to $\Sigma_{i=1,...,n}\, l_i$ do
            1. Inizialization of Candidate Nodes (see algorithm LS)
            2. Restriction
            3. Inizialization of Feasible Moves
            4. Move Selection: select a feasible move $(O_{i'j\,r'}, O_{i\,j\,r})$ of $E_j$ by means of the transition
               probability rules (6); directing the related disjunctive arc ($O_{i'j\,r'}$ ='dummy 0',if $r=1$);
            5. Arc Removing
            6. Computing length
            7. Transferring length
            8. Updating Structures
            9. Local Updating: Apply the local update rule (8) to the arcs $(O_{i'j'r'}, O_{i\,jr})$ of WDG;
        end for
        Directing the remaining disjunctive arcs
        Local Search: Apply local search routine to $S_a$;
```

```
            Best Evaluation: if ($makespan(S_a)<makespan(S_{be})$)
                               then ($makespan(S_{be}) \leftarrow makespan(S_a)$ and $S_{be} \leftarrow S_a$ )
                               end if
        end for
        Global Updating: Apply the global update rule (9);
        Best Ant Evaluation: if ($makespan(S_{be})<makespan(S^*)$)
                            then (($makespan(S^*) \leftarrow makespan(S_{be})$ and $S^* \leftarrow S_{be}$ and $epoch \leftarrow 0$);
                            else  epoch ++;
                            end if
end while
Output: $S^*$
```

4.1 Reinforced relation-learning Representational Model of Pheromone

The Relation-learning model of pheromone trail is represented by the weighted disjunctive graph $WDG \cup W_{E,p}$, where $WDG = (N_F, A, E_{h\,j}, W_N)$ (see section 3.1) and $W_{E,p}$ are the weights on the disjunctive arcs. The weight on the disjunctive arcs $(O_{i'j'r'}, O_{i\,jr})$ of $E_{h\,j}$ is represented by the 2 x n x n matrix $W_{E,p}(O_{i'j'r'},O_{ijr}) =(\tau_p[O_{i'j'r'}, O_{i\,jr}], \tau_p[O_{i\,jr}, O_{i'j'r'}])$. The first component array $\tau_p[O_{i'j'r'}, O_{i\,jr}]$ gives an index of desirability in order to insert the feasible move $[O_{i'j'r'}, O_{i\,jr}]$ in the location p of the loading sequence of resource j (p=1,..,n), in addition to the standard desirability for assigning the sub-sequence $[O_{i'j'r'}, O_{i\,jr}]$ to resource j of the standard relation-learning model.

4.2 Candidate List Restriction

The candidate list restricts the choice of the next node to visit at the construction step w to a subset of the most promising operations in the allowed list, i.e. $CL_w \subseteq AL_w$. The allowed list includes all the operations that can be selected at a given construction step of the list scheduler algorithm in order to achieve a final feasible schedule. Nevertheless, it is well-known that the optimal schedule is always an active schedule, i.e. a feasible schedule in which no operation could be started earlier without delaying some other operations or breaking a precedence constraint. Thus the search space can be safely limited to the set of all active schedules. An important class of active schedules is the Non-Delay schedule: these feasible schedules are schedules in which no resource is kept idle when it could start processing some operation. As not all optimal schedules are non-delay, the concept of *parameterized* Non-Delay schedules is used. This type of schedule consists of schedules in which no resource is kept idle for more than a predefined value δ if it could start processing some operations. As the minimum starting time of the operations in AL_w is:

$$\min_{O_{ijr} \in AL_w} st(O_{ijr}) \tag{10}$$

all the operations O^* which can start if no resource is kept idle for more than a predefined value δ, verify the following condition:

$$st(O^*) \leq \min_{O_{ijr} \in AL_w} st(O_{ijr}) + \delta, \qquad O^* \in AL_w \tag{11}$$

In this paper the following parametric value $\delta(rf)$ is used:

$$\delta(rf) = \frac{\max_{O_{ijr} \in AL_w} st(O_{ijr}) - \min_{O_{ijr} \in AL_w} st(O_{ijr})}{rf} \tag{12}$$

where rf is the restricted factor. If the restriction is maximum, i.e. $rf \rightarrow +\infty$, the predefined value $\delta(rf)$ tends to zero and we obtain a non-delay schedule; on the contrary, if rf is set to more than 0, the property of the non-delay schedule is relaxed; finally, if $rf = 0$, the candidate list does not differ from the allowed list, i.e. no restriction is achieved. To sum up, the following candidate list is used:

$$CL_W = \left\{ O^* \in AL \mid st(O^*) \leq \min_{O_{ijr} \in AL_w} st(O_{ijr}) + \frac{\max_{O_{ijr} \in AL_w} st(O_{ijr}) - \min_{O_{ijr} \in AL_w} st(O_{ijr})}{rf} \right\} \tag{13}$$

4.3 Visibility Functions

The visibility function represents the heuristic information that, together with the pheromone amount, guides the selection of the next operation in the partial schedule. It is a critical component that influences system performance. The Earliest Starting Time (EST) dispatching rule is the best function of visibility among a number of dispatching rules compared in reference (Blum and Sampels, 2004). However in this paper a novel heuristics are describes.

The first is a *static rule*, evaluated one time only at the starting of *RR-ACS*, which drastically differs from a classic dispatching rule. It is represented by a loss function obtained by comparing the starting times of a candidate operation and its lower bound obtained with a heuristic reasoning upon the routing-precedence based schedule. In the routing-precedence based schedule, all the operations which may be processed on the same resource are grouped (and processed in a sequence in a block of subsequent operations termed *layer*) on the basis of the related precedence constraint in the job-routing. A layer L_{rj} is the block of the r^{th} operations in the job routing which must be processed on a resource j; if the resources are related, the layer L_{rj} is the block which must be processed on a pool j. A lower bound for the starting time of the r^{th} operation in the job routing which must be processed on a resource of the pool j, can be evaluated by the maximum completion time of the layer $L_{r-1\ h}$, h=1,..,m, plus half of the sum of the processing times of the layer $L_{r\,j}$:

$$\begin{cases} st_j(0) = \sum_{i=1}^{n} t_{i\,j\,0} \\ st_j(r) = \max_{h=1,\dots,m}\left[st_h(r-1) + \sum_{i=1}^{n} t_{i\,h\,r} \right] + \sum_{i=1}^{n} t_{i\,j\,r} \end{cases} \tag{14}$$

If the resources are related, the average processing time on the resources of the pool replaces the sum of processing times. As a consequence of expression (12), the visibility function for an operation O_{ijr} can be defined by means of the following loss function:

$$\left[1 - \frac{t(O_{ij\,r}) - st_j(r)}{lb} \right]^2 \tag{15}$$

where lb is a lower bound for the minimum makespan. It is equal to the optimal makespan if *makespan*(S*) is known; otherwise lb can be evaluated, for example, by the maximum $st_j(m+1)$, j=1,..,m, the maximum starting time of the layer which has the precedence $m+1$. The loss function (15) ranges in [0,1]; it presents a maximum when an operation is selected such that the difference between the lower bound of the starting time and the starting time of the operation in the partial schedule is the lowest. It may be noted that in expression (15), the lower bounds for the starting times $st_j(r)$ (and in particular lb), do not depend on the starting time of the operations, and hence they are evaluated at the Initialization step of algorithm LS. The static visibility function is achieved by normalising function (15) with respect to the pheromone amount (which ranges in [τ_0, *makespan*(S^*)$^{-1}$]). Thereby, the following static visibility function is adopted:

$$\eta(O_{ijr}) = \frac{1}{lb^3}\left[lb - t(O_{ijr}) + st_j(r)\right]^2 \tag{16}$$

The second heuristic is a *dynamic rule*, evaluated for each candidate operation within the loop of *RR-ACS*. It is a more sophisticated version of the EST visibility, based on the normalized difference between the earliest starting time among the candidate operations and the starting times of each candidate operation. Thus, one values of visibility function is obtained for each candidate operation. The scale factor is the sum of the differences between the earliest starting time and the starting time of the single candidate operations.

5. Computational experiments and results

The performance of the proposed system is evaluated by benchmark problems appropriately designed for job shop scheduling with partial flexibility and separable setup times. They are obtained by modifying a set of 10 job shop scheduling problems, indicated as [$PJ_m(1)$| *prec* |C_{max}], which includes some problems considered challenging by Balas and Vazacopoulos, (1998): LA02 and LA03 (10x5), LA15 (15x5), ORB1 and ORB4 (10x10), LA21 and LA25 (15x10), LA27 and LA29 (20x10) and the well-known FT10 (10x10).

The problem instances belong to the following datasets:

i) [$PJ_m(k)$| *prec* |C_{max}]$_{k=2,\,3}$ *dataset*: 10 problems achieved by duplicating and tripling the 10 job shop scheduling problems of the [$PJ_m(1)$| *prec* |C_{max}] dataset; the paradigm to generate benchmark problems for job shop scheduling with parallel machines has been taken by Rossi and Dini (2001) where each problem involves *k* replications of each original resource and *k* replications of each original job.

ii) [$PJ_5(2)$| *prec* |C_{max}] *dataset*: 10 problems LA01'-10' which involves 2 replications (*k*=2) of each original five resources (*m*=5) and each original job of LA01-10.

iii) $[PJ_{10}(k)|\ s_j\ ,prec\ |C_{max}]_{k=1,2,3}$ *dataset* - 108 instances of the considered (general) problem derived by the famous FT10 (n=10, m=10, k=1): including its duplicate and triplicate version (k=2 and 3). For each of these, 4 scenarios are proposed; they are related to the following 4 setup activities for each resource j=1,..,10: $F_j=z$, z=1, 2, 5, 10 with $f_{ij} = i \bmod z$. They represent the problem with: a) z=1, only one resource setup at the starting of the plan, the other setup times are simply removed from processing times (no setup times); b) z=2, a low rate of sequence-dependent setup times; c) z=5, a high rate of sequence-dependent setup times; d) $z= n$, the setup times are sequence-independent; each setup can be run on the machine during the moving of the job on board the machine (in any case, before processing). The processing times are achieved by reducing by the same percentage the original times of FT10 within the range of *lag_times_rate* =10,..,40%; this reduction in time is applied to the lag times (*SI* and *SD* setup phases). *SI* activity includes maximum transportation times of 10%, 20% and 40% of the lag time (*SI_rate*).

The following metaheuristics for flexible job shop scheduling are considered for comparison:

- *GT-BDDR*: the list scheduling algorithm with the best dynamic dispatching rule used for move selection among the following: SPT, LPT, FIFO, LIFO, EST, EFT, MWR (Most Work Remaining) and Random;
- *RD-GA*: the genetic algorithm for $PJ_m(k)|\ prec\ |C_{\max}$ proposed by Rossi and Dini (2000);
- *IGA*: the current best single component metaheuristics (GA-based) for $PJ_m(k)|\ prec\ |C_{\max}$, proposed by Chan et al. (2006);
- *RR-ACS*^{1}, *RR-ACS*^{2}, *RR-ACS*^{3}, *RR-ACS*^{4}: 4 versions of the proposed ant colony system achieved by all the possible combinations of the following system components: a) standard relation-learning (*RR-ACS*^{1}, *RR-ACS*^{2}) or reinforced relation-learning model of pheromone (*RR-ACS*^{3}, *RR-ACS*^{4}); b) dynamic (*RR-ACS*^{1}, *RR-ACS*^{3}) or static visibility (*RR-ACS*^{2}, *RR-ACS*^{4}). Every other system parameter is configured by means of preliminary tests for each *RR-ACS* algorithm (i.e. β=0.3, ρ=0.12, rf=4 or 3 for, respectively, *RR-ACS*^{3} and *RR-ACS*^{4}).

Batch or *k*-decomposition is not allowed by the considered heuristics.

The $[PJ_m(k)|\ s_j\ ,prec\ |C_{max}]_{k=1,2,3}$ dataset is generated to compare the performance of the proposed system towards the general problem considered. The difficulty of optimally solving the general problem is progressively enhanced by instances with increasing number of parallel machines (*k*) and increasing rates of sequence-dependent setup times (*z*).

All the algorithms are implemented in Visual C++ 5.0 and run on Intel® Core™2 Duo, 3.1 GHz based PC. In all the experiments, except for the $[PJ_{10}(2)|\ prec\ |C_{max}]$ dataset, 5 ants (i.e. $ps = 5$) and ne=4 10^4 of allowed epochs without improvement of the best solution are adopted. Having benchmark problems, the optimality condition also includes the reaching of the optimal solution.

All the metaheuristics has been run 5 times for each benchmark, except for the rule-based heuristic *GT-BDDR*; a total of 620, 50 and 540 runs have been performed for the three considered datasets.

.

5.1 Performance comparison by means dataset described at point i)

Table 1 shows the following results obtained with the $[PJ_m(k)|\ prec\ |C_{max}]_{k=2,\ 3}$ dataset:

a) the average makespan for each of the six (meta)heuristics,

b) the minimum makespan for each of the six (meta)heuristics and

c) the average computation time of the best heuristic.

Also, the values are grouped according to the degree *k* of parallelization capability of the system giving an index of performance represented by the percent average relative error, *ARE%*, of the average makespan achieved by 5 runs on the single instance *y* (denoted by $makespan_y$), *y*=1,..,20 which is evaluated by:

$$ARE\% = 10 \cdot \sum_{Inst\ no=1}^{10} \frac{makespan_y - opt_y}{opt_y}, \quad \begin{cases} y = (2 \cdot Inst\ no) - 1 \\ y = (2 \cdot Inst\ no) \end{cases}, \quad Inst\ no = 1,..,10 \qquad (17)$$

where opt_y, is best known makespan of instance *y*.

The $makespan_y$ are showed in the same column in Table 1 (reported in normal font); *y*=(2·*Inst no*)-1 and *y*=(2·*Inst no*) are indices used to evaluate the percent average relative error obtained for the instance *Inst no*, respectively, *k*=2 and *k*=3. The same procedure is used to evaluate the percent average relative error, *ARE%*, of the minimum makespan (in italics).

The dispatching rules-based approach, *GT-BDDR*, offers a real-time response but, in general, produces high errors (*ARE%* > 22). Generally, this approach offers a similar performance compared to the number of parallel resources, *k*=2 and 3. The genetic algorithm *RD-GA* shows a similar performance compared to *GT-BDDR* for *k*=2 and the minimum makespan; slightly worse for the average makespan in the cases of *k*=3. The proposed ACOs significantly improve *GT-BDDR* and *RD-GA*. Its performance gives an *ARE%* which range (among its 4 versions) between 5.44 and 6.28, for *k*=2, and between 6.69 and 7.82, for *k*=3, which are about 3 times lower than that achieved by the compared systems. *RR-ACS*$^{1-4}$ offer a quite stable performance during the different runs: *ARE%* for the minimum makespan differs from that one of the average makespan lesser than 0.7.

On the other hand, the performance of four versions of the proposed ACO is quite different. For each row (instance), Table 1 shows in bold the minimum of the average makespan and the absolute minimum makespan obtained from six (meta) heuristics. *RR-ACS*^{3} offers the best results in 17 of 20 instances as minimum of the average makespan and in 12 of 20 instances as absolute minimum makespan. The remaining 3 best minimums of the average makespan and the 8 absolute minimums are equally divided between *RR-ACS*^{1} and *RR-ACS*^{4}.

Moreover, the following considerations can be supported by the results of Table 1:

i) In the case of *k*=2, all the different versions give *ARE%* values up to 1.8% lower than those obtained in the case of *k*=3;

ii) For *k*=2 the reinforced model of pheromone achieves the best performance; *ARE%* obtained by the dynamic visibility is less than about 0.5 (see at the bottom of Table 1 for both average and minimum makespan) compared to that one obtained by static visibility;

iii) For *k*=3 the dynamic visibility achieves the best performance; the reinforced model of pheromone obtains an increase in performance compared to the standard model; also in this case, the AREs% differ of about half point.

Instance no		*opt*	*n*	*m*	*y*	*k*	*GT-BDDR*	*RD-GA*	*RR-ACS*^{1}	*RR-ACS*^{2}	*RR-ACS*^{3}	*RR-ACS*^{4}	Time (10^{2}s)
1	LA02	655	10	5	1	2	742	759.4	686.2	687.0	**678.6**	680.2	10.1
							742	*748*	*682*	*684*	***672***	*674*	*7.1*
					2	3	745	782.8	700.4	701.2	697.4	**697.0**	47.8
							745	*768*	*696*	*699*	*693*	***692***	*32.8*
2	LA03	597	10	5	3	2	677	705.3	635.5	642.5	**627.3**	635.8	13.9
							677	*700*	*631*	*638*	***625***	*633*	*9.4*
					4	3	677	746.3	652.0	662.0	**645.5**	653.5	43.9
							677	*737*	*647*	*653*	***640***	*651*	*32.8*
3	LA15	1207	20	5	5	2	1375	1390.3	1239.5	1239.8	**1234.3**	1235.8	93.1
							1375	*1384*	*1234*	*1235*	*1231*	***1230***	*64.6*
					6	3	1411	1411.3	1253.8	1259.3	**1251.3**	1257.8	264.7
							1411	*1384*	*1249*	*1255*	***1246***	***1246***	*251.3*
4	FT10	930	10	10	7	2	1115	1207.2	978.4	980.2	**973.6**	974.8	29.3
							1115	*1187*	*974*	*977*	*969*	***963***	*22.8*
					8	3	1124	1228.2	981.2	988.6	**975.2**	988.4	120.4
							1124	*1211*	*967*	*981*	***965***	*984*	*100.9*
5	ORB1	1059	10	10	9	2	1199	1416.0	**1089.0**	1086.0	1095.8	1096.0	105.5
							1199	*1393*	***1077***	*1082*	*1092*	*1091*	*77.2*
					10	3	1177	1438.4	**1085.0**	1086.6	1098.5	1100.5	278.9
							1177	*1421*	***1081***	*1084*	*1097*	*1094*	*258.7*
6	ORB4	1005	10	10	11	2	1272	1189.6	1043.3	1040.0	**1034.0**	1036.8	23.8
							1272	*1175*	*1041*	***1027***	*1030*	*1030*	*16.0*
					12	3	1190	1214.6	1044.0	1049.8	**1033.3**	1043.3	109.8
							1190	*1196*	*1034*	*1045*	***1023***	*1040*	*85.4*
7	LA21	1046	15	10	12	2	1253	1345.0	1118.8	1118.8	**1114.0**	1117.4	142.7
							1253	*1325*	*1116*	*1111*	***1108***	*1112*	*107.8*
					14	3	1332	1363.6	1129.8	1132.6	**1126.4**	1127.6	307.1
							1332	*1348*	*1124*	*1126*	***1118***	*1122*	*288.3*
8	LA25	977	15	10	15	2	1300	1223.4	1049.0	1056.8	**1044.6**	1051.8	164.1
							1300	*1187*	***1036***	*1053*	*1037*	*1041*	*95.1*
					16	3	1329	1269.4	1068.6	1079.6	**1058.8**	1063.8	636.9
							1329	*1255*	*1063*	*1077*	***1054***	*1060*	*552.6*
9	LA27	1235	20	10	17	2	1608	1576.2	1324.8	1323.8	**1317.8**	1326.6	307.2
							1608	*1538*	*1316*	*1316*	***1302***	*1322*	*224.1*
					18	3	1619	1628.8	1332.3	1338.7	**1325.8**	1327.7	575.8
							1619	*1602*	*1324*	*1335*	***1317***	*1327*	*566.9*
10	LA29	1153	20	10	19	2	1653	1507.8	1301.8	1313.0	**1294.5**	1309.3	210.6
							1653	*1494*	*1295*	*1290*	***1280***	*1301*	*180.8*
					20	3	1594	1507.8	1322.8	1330.5	**1312.3**	1331.0	558.1
							1594	*1520*	*1315*	*1327*	***1305***	*1327*	*426.8*
ARE%							22.67	24.33	6.02	6.28	**5.44**	5.96	110.0
							22.67	*22.45*	*5.37*	*5.55*	***4.76***	*5.26*	*80.5*
							22.72	27.31	7.18	7.82	**6.69**	7.37	294.4
							22.72	*25.76*	*6.46*	*7.30*	***6.00***	*6.88*	*259.7*

Table 1 – Results achieved by using $[PJ_m(k)|\, prec\, |C_{max}]_{k=2,\,3}$ dataset. The average (minimum) makespan of 5 runs is showed in normal font (italics). Time is the computation time of $RR\text{-}ACS^3$. *ARE%* is evaluated by the expression (17) and is showed in normal font (italics) for average (minimum) makespan.

A more detailed investigation is faced by means of the non-parametric statistical Kruskall-Wallis H-test in order to determine if any algorithm performs significantly better than others. In general, the use of this test takes the place of a test for normal distribution, like Student's T test, where the value returned by the algorithm are gathered into a lower bound (i.e. the differences between the optimum values of the benchmarks and the are gathered into 0_+). The null hypothesis is that the samples returned of Table 1 are originate from the same distribution when they are grouped among the various categories of membership (k= 2, 3 and ALL, where ALL merges the categories k= 2 and k= 3).

Table 2 shows the main steps of the H test statistic for the category ALL. The null hypothesis must be rejected at the significance level of 0.05. A significant difference among the algorithms performance is very likely because for each the categories the H test values is higher than critical value at the alpha level. Having established that the null hypothesis of H test statistic should be rejected, a pair-wise comparison between the best and second best algorithm is able to explain which performs better. The Wilcoxon Signed Rank test for paired data in Table 1. The following considerations for the proposed algorithm can be supported by the results of Table 2 (bottom):

1. for each category k= 2, 3 and ALL, the best average makespan algorithm, $RR\text{-}ACS^3$, performs significantly better than its second best counterpart, $RR\text{-}ACS^1$;
2. considering both the performed test statistics, $RR\text{-}ACS^3$ offers the best performance among all the compared algorithms.

These features confirms that the proposed ACO which includes the reinforced relation-learning model of pheromone and early starting time-based visibility is superior. Without loss of generality, henceforth we will refer to $RR\text{-}ACS^3$ in terms of *RR-ACS*.

Test		*k*	*GT-BDDR*	*RD-GA*	*RR-ACS*^{1}	*RR-ACS*^{2}	*RR-ACS*^{3}	*RR-ACS*^{4}
Kruskall–Wallis H test statistic	Average rank sum (r_i/N), N=20	ALL	93.45	100.65	42.73	45.43	37.85	42.90
	(r_i –[N+1]/2)^2	ALL	1085.70	1612.02	315.95	227.26	513.02	309.76
	Ties	ALL	6	2	4	3	2	4
	Factor of correction	ALL	0.99987					
	H	ALL	**67.18**					
		2	**34.42**					
		3	**35.75**					
	Critical value for α= 0.05		**11.07**					

		k	*RR-ACS*^{3}	*RR-ACS*^{1}
Wilcoxon Signed-Rank Test for paired data	Rank sum (R_i)	ALL	31	177
	W	ALL	146	
	σ	ALL	53.57	
	Z	ALL	**2.72**	
		2	**2.22**	
		3	**2.32**	
	Z critical for α= 0.05		**1.960**	

Table 2 – Non-parametric Kruskall–Wallis H test statistics for algorithms comparison and Wilcoxon Signed-Rank Test for paired data returned by the two best proposed ACO configurations. Analisys of data in Table 1 ($[PJ_m(k)|\ prec\ |C_{max}]_{k=2,3}$ dataset) in the cases of: i) All data; ii) *k*=2 only, iii) *k*=3 only.

These features confirms that *RR-ACS*^{3} , the proposed ACO which includes the reinforced relation-learning model of pheromone and early starting time-based visibility, offers the best performance among all the compared systems. Without loss of generality, henceforth we will refer to *RR-ACS*^{3} in terms of *RR-ACS*.

5.2 Performance comparison with literature method

The large difference in performance with *GT-BDDR* and *RD-GA* does not allow to give an objective evaluation of the proposed system. Therefore, the performance of the proposed system is compared with the *IGA*, the algorithm proposed by Chan et al., 2006 which currently obtains the best results for $PJ_m(k)|\ prec$ $|C_{\max}$. For this purpose the $[PJ_5(2)|\ prec\ |C_{max}]$ dataset (described at point ii)) is adopted.

As the data size of the [$PJ_5(2)|$ *prec* $|C_{max}$] dataset is lower than the other datasets, *RR-ACS* has been run with a higher computation power (*ps*=12) and a stability condition less time consuming: the number of epochs allowed without obtaining an improvement of the best solution is reduced by an order of magnitude compared to the previous experiment ($ne=10^3$).

RR-ACS has been run 5 times. Table 3 shows the best makespan ($RR\text{-}ACS_{min}$), the average makespan ($RR\text{-}ACS_{ave}$) and the average computation time achieved by *IGA* and *RR-ACS*. *IGA* solved to the optimality 3 of the 10 instances. The proposed ACO solves the optimality 6 instances, of which 4 of them (LA05', LA06 ', LA09 'and LA10') in all 5 executions. The others 2 instances (LA01' and LA08') are solved to the optimality, respectively, in 4 and 3 executions with an average relative error lower than 0.05%.
Four cases are still open. However, in one of these open instances (LA07') the percent relative gap, *D2%*, is widely lower than 1%.
RR-ACS improves the performance obtained by *IGA* in 6 of 10 instances (LA01', LA02', LA03', LA04', LA05' and LA09'), i.e. has been found a lower average makespan $RR\text{-}ACS_{ave}$.
The percent relative gap $D1\%=(IGA\text{-}RR\text{-}ACS_{ave})/IGA$ x 100, is also considered for algorithm comparison; the gap $D2\%=(opt\text{-}ACS_{ave})/opt$ x 100, is used to compare *RR-ACS* with the optimal solution.
RR-ACS achieves a significant percent relative improvement, *D1%*, which ranges from 1.16% and 2.16%, in 3 instances (LA02', LA03' and LA04') where *D2%* now ranges from 2.6 and 3.9%. In others 2 instances (LA06' and LA10') achieved the same performance in terms of average makespan (hence also the best makespan) considering that *IGA* was run only one time. Only one instance solved to the optimality by *IGA* (LA08'), *RR-ACS* fails 2 times on 5 executions.
Finally, in one instance (LA07') *RR-ACS* is not able to outperform the competitor algorithm. However, the percent relative gap, *D2%*, is widely lower than 1%.
The claims made are supported by the non-parametric statistical test for paired data of the two independent samples of Table 3, showed the bottom of same table. The Wilcoxon Signed Rank test statistic in the pair-wise comparison with IGA shows that a significant difference between the algorithms performance is probable because the overall difference observed between the two samples is significant up to the alpha level of 0.1 and the Z test statistic is very close to 95% of the confidence interval.

Inst.	opt	*n*	*m*	*k*	*IGA*	*RR-ACS*	*D1%*	*D2%*	*IGA* time (10^2s)	*RR-ACS*[3] time (10^2s)
LA01	666	10	5	2	668	**666** (666.2)	0,27	0	3,5	0,9
LA02	655	10	5	2	692	672 (684.0)	1,16	-2,60	3,5	1,3
LA03	597	10	5	2	637	619 (627.8)	1,44	-3,69	3,7	1,1
LA04	590	10	5	2	629	613 (615.4)	2,16	-3,90	3,5	1,5
LA05	593	10	5	2	595	**593** (**593.0**)	0,34	0	3,6	0,0
LA06	926	15	5	2	**926**	**926** (**926.0**)	0	0	4,9	0,2
LA07	890	15	5	2	891	895 (898.0)	-0,79	-0,56	5,2	3,6
LA08	863	15	5	2	**863**	**863** (863.4)	-0,05	0	4,8	1,5
LA09	951	15	5	2	954	**951** (**951.0**)	0,31	0	4,7	0,7
LA10	958	15	5	2	**958**	**958** (**958.0**)	0	0	5,0	0,2
Rank sum (R_i)					10	44				
W					34					
σ					19.62					
Z					**1.71**					
Z critical for α= 0.05					**1.960**					
Z critical for α= 0.1					**1.6448**					

Table 3 – Metaheuristics comparison by using [$PJ_5(2)|$ *prec* $|C_{max}$] dataset and and Wilcoxon Signed-Rank Test for paired data. IGA stands for Iterative Genetic Algorithm (Chan et al., 2006). The column *RR-ACS* shows the minimum ($RR\text{-}ACS_{min}$) and the average ($RR\text{-}ACS_{ave}$, in round brackets). $D1\%=(IGA\text{-}RR\text{-}ACS_{ave})/IGA$ x 100%; $D2\%=(opt - RR\text{-}ACS_{min})/opt$ x 100%.

The average computation time of *RR-ACS* is lower than 100 s for each instance solved to the optimality (except for LA08'). In general, the average time spent is lower than 150 s (except for LA07') making the algorithm stable.

RR-ACS considerable faster than *IGA*. In fact, despite the fact that *RR-ACS* has been tested on a faster computer, the time spent is lower than *IGA* of about 4 times. In particular, the time spent to solve to the optimality the four aforementioned instances in all the executions is at least 7 times lower than that one of *IGA*.

In conclusion, for both the considered indices of performance (makespan and computation time) *RR-ACS* is superior to the competitor single component metaheuristics (ACO-based vs GA-based) also, for the sake of completion, the performance is similar to that one of the two-component metaheuristics (GA and ACO) proposed by the same authors (Rossi and Boschi, 2009).

5.3 General problem: performance and discussion

In the general problem, the performance of *RR-ACS* is compared with the lower bound for the minimum makespan, *lb*, achieved by means of the following expression:

$$lb_y = opt_y(1 - lag\ time\ rate)\left\{\min_{O_{IJ0},z^*=f_{IJ}} t(f_{ij}) + \sum_{\substack{z=1 \\ z\neq z^*}}^{F_j} \min_{O_{IJ0},z=f_{IJ}} t(f_{ij})\right\} \qquad (18)$$

Expression (18) considers the sum of the minimum setup times for each kind of setup F_j. As z is the number of different kind of setup, *lb* is evaluated considering exactly z setup changes, i.e. the operations whose have the same kind of setup activity are grouped in the machine sequence.

As anticipated, has been used 108 instances of $[PJ_{10}(k)|\ s_j\ ,prec\ |C_{max}]_{k=1,2,3}$ dataset derived from FT10 (instance no.7 in Table 1, where $opt_7 = 930$). The results obtained are analyzed in detail by extending the sole value achieved from FT10 to a sample of 10 observations (independent) with different number of machines and jobs. The samples are obtained like for $[PJ_{10}(k)|\ s_j\ ,prec\ |C_{max}]_{k=2}$ dataset considering the instances of Table 1 (i.e. LA02, LA03, LA15, FT10, ORB1, ORB4, LA21, LA25, LA27 and LA29). Each sample of 10 observations belongs to one of the 36 category of number of setup, lag time rate and SI rate, having limited samples at k=2. This restriction is no loss of generality because, as shown below, the characteristics of the curves for k=2 and k=3 are not dissimilar.

The computational time to perform this extension has been strongly reduced. *RR-ACS* uses a stability condition about by one order of magnitude (i.e. ne=10^3)."

The performance are summarized in Table 4 and Figure 3.

	← z (=F_j, number of setup) →			
	1	2	5	10
lag time rate	← *SI_rate* (rate of transportation time within lag time) →			

		10	20	40	10	20	40	10	20	40	10	20	40
	lb	*841.1*	*840.7*	*839.8*	*846.6*	*845.2*	*843.2*	*857.7*	*855.4*	*850.8*	*886.3*	*880.8*	*869.9*
10	*k*=1	895.1	899.6	907.0	925.5	918.2	941.1	939.9	941.4	944.0	946.5	944.9	947.0
	k=2	909.5	907.2	917.7	928.0	927.6	919.8	931.6	940.9	941.9	953.9	952.1	952.8
	k=3	932.0	932.1	929.8	945.8	944.7	955.6	958.3	952.6	967.8	963.8	962.8	966.7
	lb	*752.3*	*751.4*	*749.5*	*762.5*	*760.5*	*756.4*	*785.4*	*780.8*	*771.6*	*842.6*	*831.7*	*809.8*
20	*k*=1	808.7	817.0	826.6	828.6	848.4	853.3	893.4	875.2	875.2	902.7	879.4	887.7
	k=2	818.2	828.4	841.2	850.1	865.4	870.6	886.7	894.5	900.7	907.4	916.5	909.3
	k=3	827.2	848.2	859.5	866.4	877.2	893.3	909.9	908.5	899.8	914.4	923.6	907.0
	lb	*574.6*	*572.7*	*569.0*	*595.1*	*591.0*	*582.7*	*640.8*	*631.6*	*613.2*	*755.3*	*733.4*	*689.5*
40	*k*=1	637.6	653.8	657.7	693.7	704.3	752.6	781.8	785.4	777.5	810.5	802.3	791.0
	k=2	631.6	649.3	681.2	719.3	729.5	748.1	796.0	798.9	800.2	833.6	818.8	813.5
	k=3	646.4	663.0	706	746.2	753.2	766.2	816.2	819.8	825.6	832.9	815.1	812.2

Table 4 – Minimum makespan of the proposed system in comparison with the lower bound (18) considering 108 benchmark problems of the [$PJ_{10}(k)$| s_{jk} ,*prec* |C_{max}]$_{\text{k=1,2,3}}$ dataset where different rates of resource flexibility, separable transportation and sequence-dependent setup times are considered.

Table 4 shows *lb* and *RR-ACS*$_{min}$ (the minimum makespan achieved by 5 runs) for each instance of the [$PJ_{10}(k)$| s_j ,*prec* |C_{max}]$_{k\text{=1,2,3}}$ dataset. The rows of the Table 4 are ordered for increasing values of the number of parallel machines. Increasing *k* and the rates of sequence-dependent setup times (*z*=1 vs *z*=5) the instances become more harder.

A more comprehensive analysis of the proposed system considers the relative gap *D2* as performance measure on the [$PJ_{10}(k)$| s_j ,*prec* |C_{max}]$_{k\text{=1,2,3}}$ dataset where the optimal solution *opt* is replaced with the lower bound *lb*. The average relative error versus lower bound obtained by a state-of-art metaheuristics gives also a sort of hardness of the problem instance to be solve.

Figure 3 and 4 shows the results; they are stratified with respect the number of parallel machines (*k*=1,2 and 3), the minimum number of setup changes (z=F_j =1, *j*=1,..,5,), the lag time rate removed by the processing time of the original instance FT10 (*lag time rate*=10, 20 and 40%), the transportation time rate removed by the lag time rate (*SI rate*=10, 20 and 40%) and the computation time (*RR-ACS* time) for increasing values of parallel machines in each group.

Figure 3 shows the results related to the different number of setups (showed on X-axis) and lag time rates (showed with different colours) versus the number of parallel machines (showed with different line stiles); each value is the average of all the values with the same number of setup changes, lag time rate and parallel machines number.

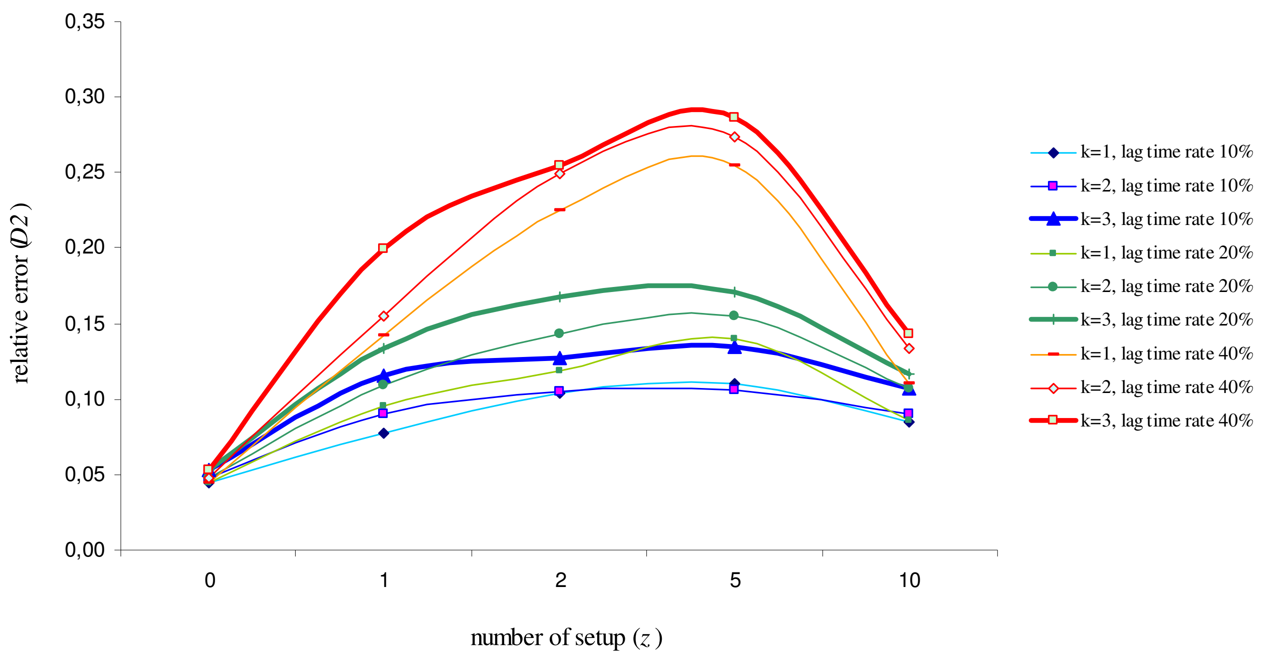

Figure 3 –Average relative error of *RR-ACS* obtained varying the number of setups for each of the 9 categories of *k* and *lag time rate* (k=1, 2 and 3. *lag time rate* =10, 20 and 40%), obtained by grouping the 3 values of *SI rate* (10, 20 and 40%). The abscissa *z*=0 represents the classical-benchmark FT10 where setup and transportation time are not considered; it is associated to 3 values obtained for *k*=1, 2 and 3.

For completeness, Figure 3 shows the value *z*=0 that represents the classical-benchmark FT10 where setup and transportation time are not considered. The relative error obtained for *z*=0 is the minimum and differs only slightly for different values of *k*.

When it is introduced a single initial setup for each machine, (*z*=1), the impact of only the transport time is assessed. For example, this is the case of the parts which are of the same family and that require a single machine setup. In this case, the makespan obtained is lower than that achieved for sequence-dependent setup times. This evidence is obtained for both high and low lag time rates as showed in Figure 4. The statistical test for paired data of the samples achieved by *z*=1 and *z*=2 for both the lag time rates of 10% and 40% (30 observations each: three SI rates per each instance Inst no, Inst no=1,..,10) shows that a significant difference between the samples at the alpha level of 0.05. The same applies to the Z-test for the samples achieved by one and five setups.

Figure 4 also shows that Z-test rejects the null hypothesis for the samples achieved by two and five setups, for both the lag time rates of 10% and 40%. The reason is that increasing the number of setups (*z*=1 vs *z*=2 and *z*=2 vs *z*=5), the number of alternative sequences to be evaluated increases exponentially and the system fails to minimize the number of setups in some sequence of machine, so that the impact on the solution found is worse when setup times are greater. In other words, the problem with sequence-dependent setup times become much more difficult to solve. In particular, the maximum gap is obtained with high rate of sequence-dependent setup times and it ranges from about 1%, for lag time rate of 10%, to 15% and over, for lag time rate of 40%. In fact, the impact of a wrong decision that occurs in the choice to perform or do not perform a setup in a machine sequence (cases *z*=2 and 5) is emphasized by having a high setup time.

For the same causes, increase the lag time rate leads to worse solutions. In all the cases *k*=1, 2 and 3 of Figure 3, the proposed system obtains lower makespan when low rates of sequence-dependent setup times are introduced, i.e. the curves of blue, green and red that occupy increasing ranges of relative error. This fact is confirmed by a more detailed analysis which allows to reject the null hypothesis, i.e. the samples obtained for different lag time rates (within the same number of setups) belongs to the same distribution. Figure 5 shows the Z test is significant at alpha level of 0.05, except for lower rate of setup times (*z*=1). The relative error linearly increases when the lag time rate increases and, on average, the maximum error is about 30%. To decrease the setup time introducing modular fixture elements which allows their (partial) reusability during setup changes is mandatory.

.

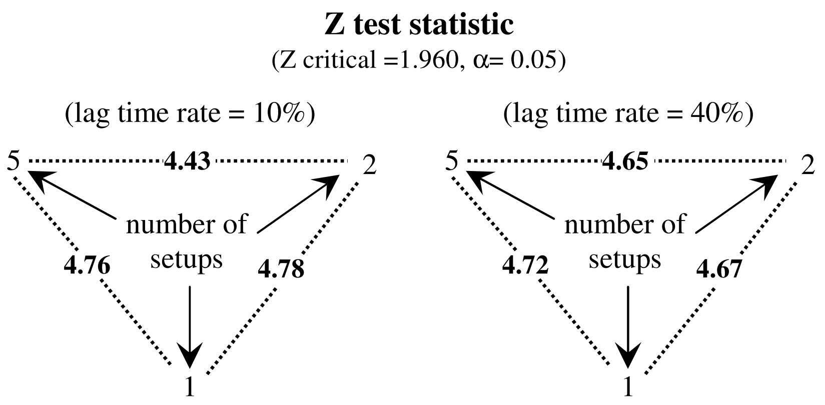

Figure 4. Wilcoxon Signed-Rank Test for paired number of setups (z_1, z_2) with z_1, $z_2 \in \{1, 2, 5\}$, $z_1 \neq z_2$, for both the lag time rates of 10% and 40%. All the Z test statistics (in bold) are higher than Z critical at the alpha level of 0.05.

Z-test of Figure 4 makes evidence that increasing the number of setups, the scheduling problem become more difficult. However, by applying the same test statistic to the samples achieved by five and ten setups (for both the lag time rates of 10% and 40%) there is no significant difference in performance. The Z values are respectively, 1.58 and 1.94. The fact that the proposed system shows the same performance in order to solve the sequence-dependent compared to the sequence-independent setup times scheduling problem means that the proposed system is able to work to the best because the first is intrinsically more difficult to solve. A possible justification is that by introducing the transportation of jobs, a twice number of operations will be introduced. Therefore, the scheduling problem with sequence-independent setup and transportation times becomes more difficult than the one with no transportation times.

Finally, when sequence-independent setup times are considered (z=10), the system performance is not comparable with that one of the first case (z=1) because all jobs require setup before processing each operation.

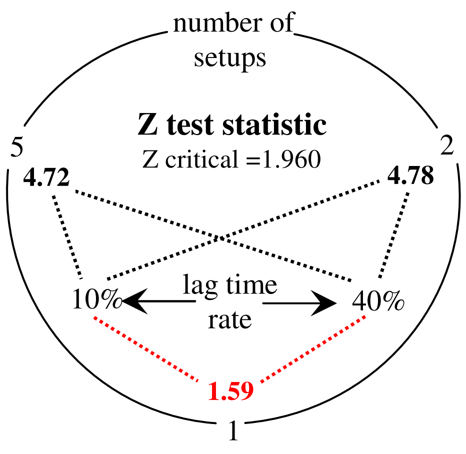

Figure 5. Wilcoxon Signed-Rank Test for paired lag time rates (10%, 40%), for each of the three cases of the number of setups: z=1, 2 and 5. The Z critical value for α=0.05 is 1.960. Bold black (red): null hypothesis rejected (accepted).

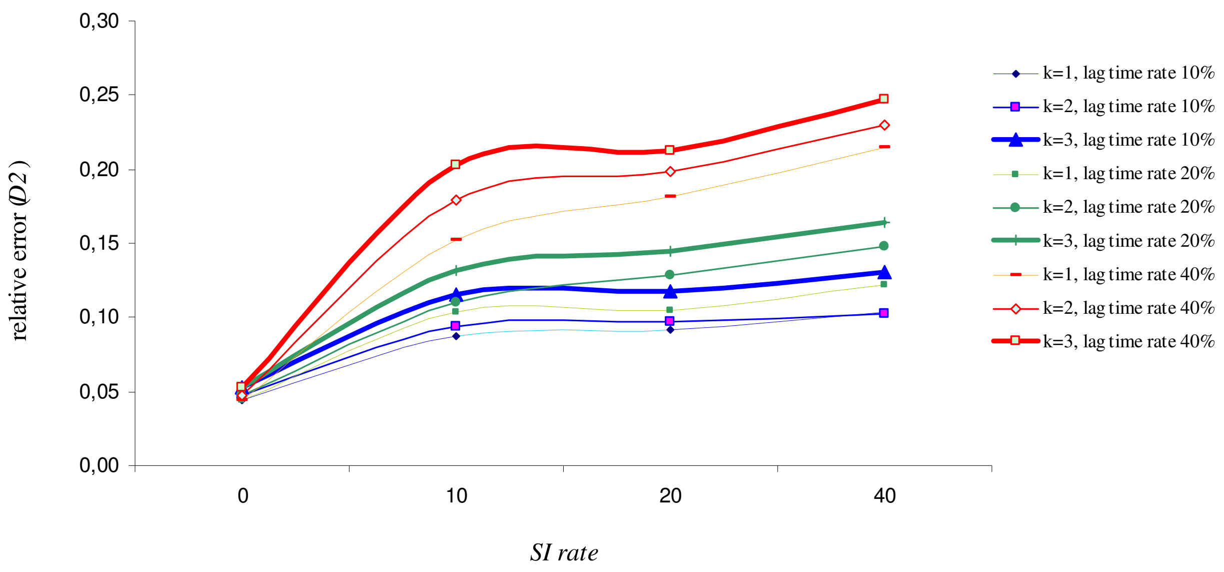

Figure 6 – Average relative error of *RR-ACS* varying *SI rate* for each of the 9 categories of *k* and *lag time rate* (*k*=1, 2 and 3. *lag time rate* =10, 20 and 40%), obtained by grouping the 4 values of number of setup (*z*=1, 2, 5 and 10). The abscissa *SI rate*=0 represents the classical-benchmark FT10 where setup and transportation time are not considered.

Figure 6 shows the performance related to the different rates of time to move the jobs among the machines (i.e. *SI rate*, showed on X-axis) included in the overall rate of lag time (showed with different colours) versus the number of parallel machines (showed with different line stiles); each value is the average of all the values with the same *SI rate*, lag time rate and parallel machines number. Even here, the value 0 represents the classical benchmark FT10 where setup and transportation time are not considered.

The proposed system offers the same performance for different values of *SI rate*, except when *SI rate* are high especially for high *lag time rate*. The H-test among the samples achieved for *SI rate* of 10%, 20% and 40% (30 observations each) leads to accept the null hypothesis for each value of number of setups, *z*=1, 2, 5 and 10 (Table 5). In contrast, the Z-test in Table 6 leads to reject the null hypothesis when the compared samples are achieved by *SI rate* of 10% and 40% for the maximum considered *lag time rate* (30 observation each, having merged *z*=1, 2 and 3).

	← z (=F_j, number of setup) →			
	1	2	3	4
H	1.12	0.68	0.86	0.95
H critical for α= 0.05	5.991			

Table 5. H test statistics for comparing different SI rates (*SI rate*=10%, 20% and 40%) for each number of setup, z = 1,2,5 and 10.

Therefore substantially, the harder instances of the problem at hand are that ones which include high *lag time rate* and *SI rate*. Meanwhile, by Z-test of Figure 5 we observed that increasing the lag time rate leads to worse solutions. Now, the Z-test in Table 6 between the samples (*lag time rate*, *SI rate*)=(10%, 10%) and (40%, 40%), having merged all the considered lag time rates (30 observations) leads us to reject the null hypothesis. This means that the proposed system works to the best when transportation and sequence-independent setup times are low.

Pairs of (lag time rates, SI rate)	(40%, 40%) versus (40%, 10%)	(40%, 40%) versus (10%, 10%)
Z	2.21	2.83
Z critical for α= 0.05	1.960	

Table 6. Wilcoxon Signed-Rank Test for the following pairs of lag time rates and SI rate: (i) (10%, 40%) vs (40%, 40%); (ii) (10%, 10%) vs (40%, 40%). Both (i) and (ii) consider data achieved by merging all categories of number of setups (*z*=1, 2 and 3).

It can be noticed that the impact of *SI rate* on the number of setups is lesser than that one of Figure 3, related to the overall rate of lag time. Figure 7 shows this behavior. In particular, the form of the relative error is the same, but the curves are shifted downwards and are all thinned in a smaller error range. Differently of the diagram of number of setup versus lag time rate (Figure 3), the relative error differs little when the *SI* rate increases and the maximum error is about 20%. This means that the marginal component of error due to *SI* rate is dominated by the other two: rate of sequence-dependent setup time and lag time.

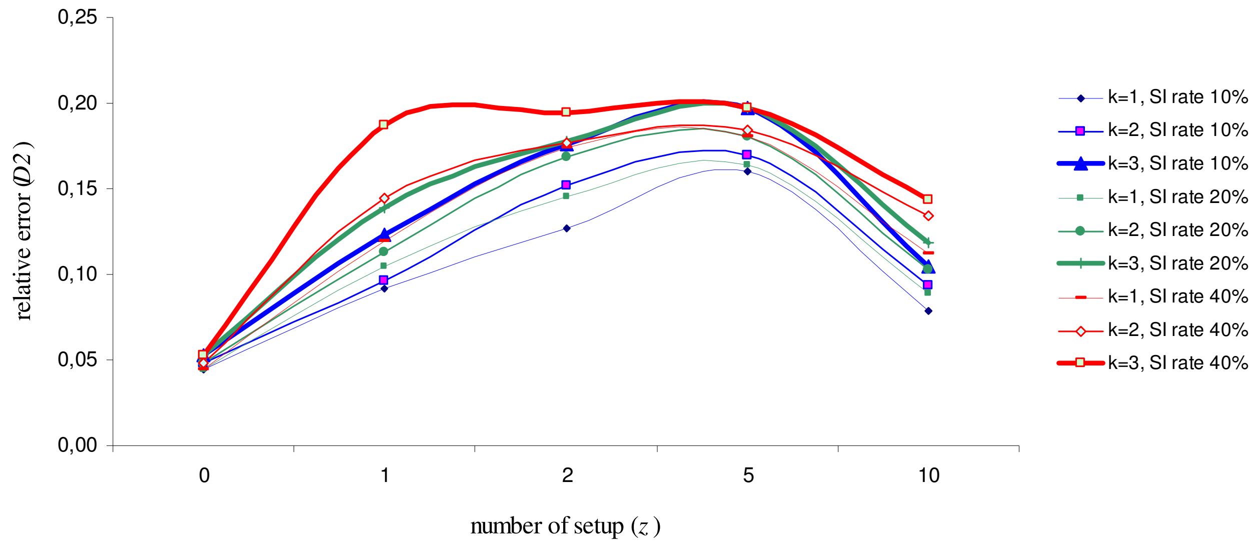

Figure 7 –Average relative error of *RR-ACS* obtained varying the number of setups for each of the 9 categories of *k* and *SI rate* (k=1, 2 and 3. *SI rate*=10, 20 and 40%), obtained by grouping the 3 values of *lag*

time rate (10, 20 and 40%). The abscissa *z*=0 represents the classical-benchmark FT10 where setup and transportation time are not considered.

The performed statistical tests confirm that obtained with the dataset derived from FT10. Similar tests can be used for *k*=3, and then verify that shown by Figures 3 and 4: i) all the 15 combinations of number of setup and lag time rate and ii) for all the 15 combinations of lag time rate and SI rate; the related error increases at most linearly with the number of parallel resources included in each group. The deviation among the diagrams for different values of *k* are very close because the maximum deviation does not exceed 5%. This fact makes the system very robust in order to approach both the assignment problem and the exponentially increasing of the setup sequences within a loading sequence.

Another interesting feature of the system is the CPU time, showed in Figure 8. Considering the minimum number of parallel machines (*k*=2) in comparison with the case with no routing flexibility(*k*=1), the CPU time offers a sharp of increasing (of about 10 times). For *k*=2, CPU time is on average 10^3 s (with a maximum of 1.23 10^3 s) while for *k*=1 it is a constant of about 10^2 s. This behaviour deals with the increase in complexity of the classical job-shop problem when the further degree of freedom related to the assignment problem has to be considered. Increasing the rate of routing flexibility, the system faces quite well the assignment to alternative resources. On average, for *k*=3, the CPU time is $2\cdot10^3$ s, which increases up to $2.53\cdot10^3$ s in the worst case (*z*=10, *lag time rate*=10). The percentage increase is about 50% compared to *k*=2. In general, the CPU time shows an exponential increase of 1.38 to approach the problem for increasing rates of routing flexibility.

The CPU time trend shows a growth trend for increasing rates of sequence-dependent setup times (*z*=1, 2 and 5). Finally, the proposed ACO converges more slowly when the lag time rate increases (except for in one case for *k*=3). In fact, changes in machine sequences little impact on the structure of the optimal sequence where setup or transportation times are not comparable with processing time. This means that the proposed ACO is more robust in advanced manufacturing systems where material-handling system is immediately available when a job completes, the time to move parts in the system is minimized and modular fixture elements allow to decrease the setup times.

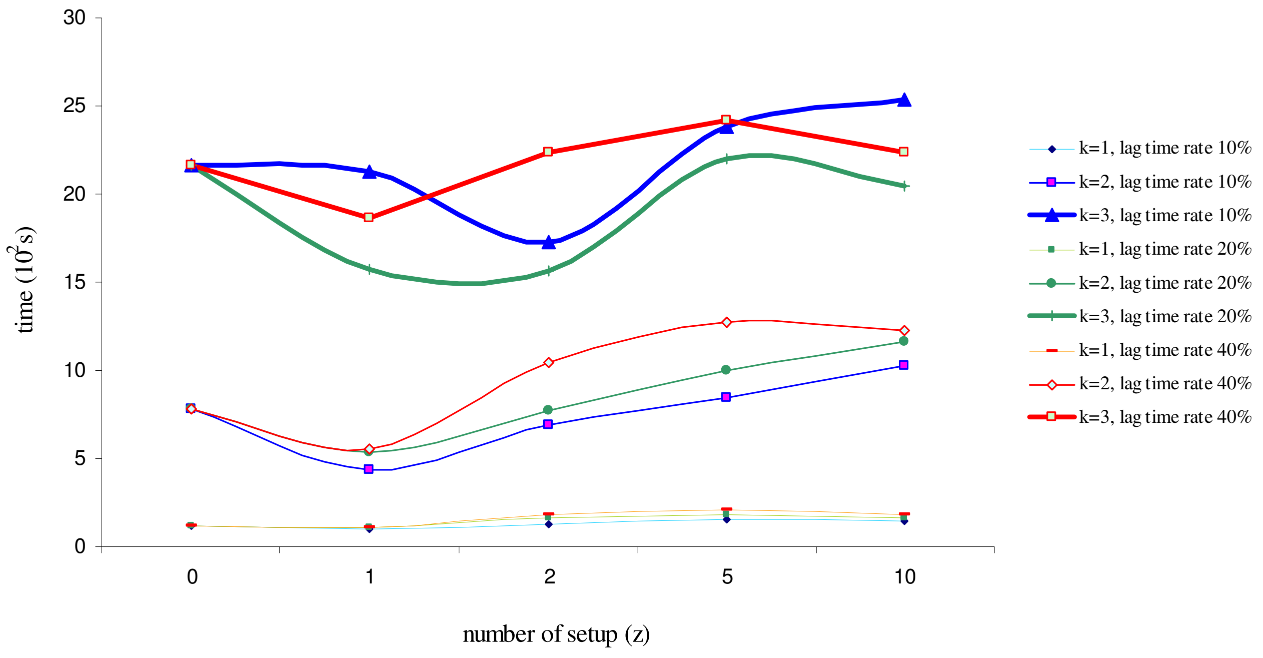

Figure 8 – CPU times (10^2s) obtained varying the number of setups for each of the 9 categories of *k* and *lag time rate* (k=1, 2 and 3. *lag time rate* =10, 20 and 40%), obtained by grouping the 3 values of *SI rate* (10, 20 and 40%). The abscissa *z*=0 represents the classical-benchmark FT10 where setup and transportation time are not considered.

7. Conclusions

The proposed ant colony optimization is a challenging approach to the job shop scheduling with a number of considered features: alternative resource, sequence-dependent setup and transportation problem. Some innovative skills are considered. The system is based on the disjunctive graph model and a list scheduler algorithm. They are able to support lag times and integrate them with the selection of alternative resources per operation. The ant colony optimization is based on a disjunctive graph where a reinforced relation-learning model of the pheromone is implemented. New tools which combines heuristic desirability (routing-precedence based and earliest starting time visibility) with a method to approximate non-delay schedules are introduced to improve the performance of the ant colony system.

As particularly stressed, statistical tests show that the reinforced relation-learning model of pheromone performs better than all the alternative configurations where the relation-learning model is not reinforced with the constraint on the job position in the machine sequence. The best configuration also includes the function of dynamic visibility, obtained by modifying the earliest starting time rule. The system performs better than other tested dispatching rules-based and genetic algorithms as shown by non-parametric test statistics. In particular, the proposed system performs significantly better that the current state-of-art algorithm (for a problem more simplified of that one considered) up to the alpha level of 0.1; so the proposed system is superior to approach the problem at hand.

In order to solve the assigning and sequencing sub-problems with a number of alternative (parallel) resources, the system performance seems quite close to that one obtained for the classic job shop scheduling and CPU time shows an exponential increase slightly higher than 1.

Experimental results and statistical tests show that the proposed system is able to work to the best. The system faces quite well the job shop scheduling with sequence-dependent and sequence-independent setup times. It shows no significant difference in performance in order to schedule the sequence-independent compared to the sequence-dependent setup times with a medium rate of setups.

The system is no significantly influenced by low setup times. At the other hand, his performance is not influenced by transportation times, except when this time is high especially for high lag time rate (considering transportation and setup). Therefore, the system is more robust and performs better in advanced manufacturing systems, where setup and transportation times have become very low by highly automated systems and tools for handling and moving parts.

Future work will be directed to increasing computing speed for real-time response behavior in multi-mode dynamic scheduling applications where the routing flexibility is extended from parallel resources to alternative process plans.

References

Allahverdi, A., J.N.D. Gupta & T. Aldowaisan, A survey of scheduling problems with setup times or costs. European Journal of Operational Research 2008;187:985-1032.

Artigues, C. & F. Roubellat, A Petri net model and a general method for on and off-line multi- resource shop floor scheduling with setup times. International Journal Production Economics 2001;7:63-75.

Balas, E. & A. Vazacopoulos, Guided Local Search with Shifting Bottleneck for Job Shop Scheduling. Management Science 1998;44:262-275.

Blazewicz, J., W. Domschke & E. Pesch, The Job Shop Scheduling Problem: Conventional and New Solution Techniques. European Journal of Operational Research 1996;93:1-33.

Blum, C. & M. Sampels, An Ant Colony Optimization Algorithm for Shop Scheduling Problem. Journal of Mathematical Modelling and Algorithms 2004;3:285-308.

Bonabeau, E., M. Dorigo & G. Theraulaz, Inspiration for optimization from social insect behaviour. Nature 2000;406:39-42.

Brucker, P. & R. Schlie, Job-Shop Scheduling with Multi-Purpose Machines. Computing 1990;45:369 – 375.

Brucker, P. & O. Thiele, A branch & bound method for the general-shop problem with sequence dependent setup-times. Operation Research Spektrum 1996;18:145-61.

Chan, F. T. S., T. C. Wong, & L. Y.Chan, Flexible job-shop scheduling problem under resource constraints. International Journal of Production Research 2006, 44(11), 2071-2089.

Colorni, A., M. Dorigo, V. Maniezzo & M. Trubian, Ant system for job-shop scheduling. Belgian Journal of Operation Research, Statistics and Computer Science 1994;34:39 – 53.

Dauzère-Pérès, S. & J. Paulli, An Integrated Approach for Modelling and Solving the General Multiprocessor Job-Shop Scheduling Problem using Tabu Search. Annals of Operation Research 1997;70:281-306.

Dauzère-Pérès, S., W. Roux & J.B. Lasserre, Multi-Resource Shop Scheduling with Resource Flexibility. European Journal of Operational Research 1998;107:289-305.

De Jong, K.A. & W.M. Spears, Using genetic algorithm to solve NP-complete problems. Proceedings of the Third International Conference on Genetic Algorithms 1995:124–32. San Mateo, CA: Morgan Kaufmann.

Dorigo, M. & L.M. Gambardella, Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transaction on Evolutionary Computation 1997;1:53-66.

Proceedings of the Third International Conference on Genetic Algorithms 1995:124–32. San Mateo, CA: Morgan Kaufmann.

Garey, M.R., D.S. Johnson & R. Sethi, The complexity of the flowshop and jobshop scheduling. Mathematics and Operation Research 1976;1:117-129.

Giffler, D. & G.L. Thompson, Algorithms for solving production scheduling problems. Operation Research 1960;8:487–503.

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

Hurink, J., B. Jurisch & M. Thole, Tabu Search for the Job Shop Scheduling Problem with Multi-Purpose Machine. Operation Research Spektrum 1994;15:205-215.

Hurink, J. & S. Knust, Tabu Search Algorithms for Job-Shop Problems with a Single Transport Robot. European Journal of Operational Research 2005;162:99-111.

Ivens, P. & M. Lambrecht, Extending the shifting bottleneck procedure to real-life applications. European Journal of Operational Research 1996;90:252-268.

Jain, A.S. & S. Meeran, Deterministic Job Shop Scheduling; Past, Present and Future. European Journal of Operation Research 1999;113:390-434.

Kacem, I., S. Hammadi & P. Borne, Approach by Localization and Multiobjective Evolutionary Optimization for Flexible Job-Shop Scheduling Problems. IEEE Transactions on Systems, Man, and Cybernetics, Part C 2002;32:1-13.

Kumar, R., M.K. Tiwari, & R. Shankar, Scheduling of Flexible Manufacturing System: An Ant Colony Optimization Approach. Journal of Engineering Manufacture – Part B 2003;217:1443-1453.

Mastrolilli M. & L.M. Gambardella, Effective Neighbourhood Functions for the Flexible Job Shop Problem. Journal of Scheduling 2000;3:3-20.

Nowicki, E. & C. Smutnicki, A Fast Taboo Search Algorithm for the Job Shop Problem. Management Science 1996;42:797-813.

Rossi A. & G. Dini, Dynamic Scheduling of FMS Using a Real-time Genetic Algorithm. International Journal of Production Research 2000;38:1-20.

Rossi A. & G. Dini, An evolutionary approach to complex job-shop scheduling and flexible manufacturing system scheduling. Journal of Engineering Manufacture – Part B 2001;215:233-245.

Rossi A. & G. Dini, Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. Robotics and Computer-Integrated Manufacturing 2007;23:503-16.

Roy, B. & B. Sussmann, Les Problèmes d'Ordonnancement Avec Contraintes Disjonctives. Technical report, DS No. 9 bis, 1964, SEMA, Montrouge.

Stecke, K.E. & N. Raman, FMS Planning Decision, Operating Flexibilities and System Performance. IEEE Transaction on Engineering Management, 1995;42:82-90.

Stutzle, T. & H.H. Hoos, MAX-MIN Ant System. Future Generation Computer System 2000;16:889-914.