Authors' accepted manuscript (post print)
Rossi, A., & Lanzetta, M. (2012). Featuring native metaheuristics for non-permutation flowshop scheduling. *Journal of Intelligent Manufacturing*. 25(6): 1221-1233.
https://doi.org/10.1007/s10845-012-0724-8

Native Metaheuristics for Non-Permutation Flowshop Scheduling

Andrea Rossi, Michele Lanzetta
Department of Civil and Industrial Engineering, University of Pisa, Italy

Abstract

The most general flowshop scheduling problem is also addressed in the literature as non-permutation flowshop (NPFS). Current processors are able to cope with the $(n!)^m$ combinatorial complexity of non-permutation flowshop scheduling by metaheuristics. After briefly discussing the requirements for a manufacturing layout to be designed and modeled as non-permutation flowshop, a disjunctive graph (digraph) approach is used to build native solutions. The implementation of an Ant Colony Optimization (ACO) algorithm has been described in detail; it has been shown how the biologically inspired mechanisms produce eligible schedules, as opposed to most metaheuristics approaches, which improve permutation solutions. ACO algorithms are an example of native non-permutation (NNP) solutions of the flowshop scheduling problem, opening a new perspective on building purely native approaches. The proposed NNP-ACO has been assessed over existing native approaches improving most makespan upper bounds of the benchmark problems from Demirkol *et al.* (1998).

Keywords: Manufacturing systems; Flow Line; Ant Colony System (ACS); NPFS benchmarks

1 Introduction

In a manufacturing system, parts (jobs) visit machines based on a production plan (or schedule). The scheduling problem has the purpose of optimizing resources (e.g. machines utilization, workers time etc.). A general solution is not yet available and hence heuristics and metaheuristics solutions are proposed and compared on given problem benchmarks in order to achieve better schedules and consequently economical benefits.

Scheduling solutions for a manufacturing system model can be scaled in many senses: a job can be a part, the whole product or a batch; machines (or stages) can be a single operating unit, a cell, a line, or their combinations; and time is measured by non-dimensional units (e.g. from seconds to months). Examples of flow lines include transfer lines (assembly, machining), continuous processes (steel-making, heat treatments, coatings, chemical plants), and are available outside the manufacturing environment (logistics, computer science, services, with high capacity utilization, Rossi *et al.*, 2012).

The flowshop scheduling problem occurs whenever it is necessary to schedule a set of *n* jobs on *m* machines so that each job visits all machines in the same order. In the literature, there are two major types of flowshop scheduling. In a *permutation* flowshop (PFS) the sequence jobs visit machines (*routing*) is the same for all jobs, and machines process all jobs with the same sequence (or permutation). On the contrary, in a *non-permutation* flowshop (NPFS) all jobs visit all machines in the same sequence, but passing of jobs on machines is allowed, i.e. the sequence (or permutation) of jobs can be different on subsequent machines.

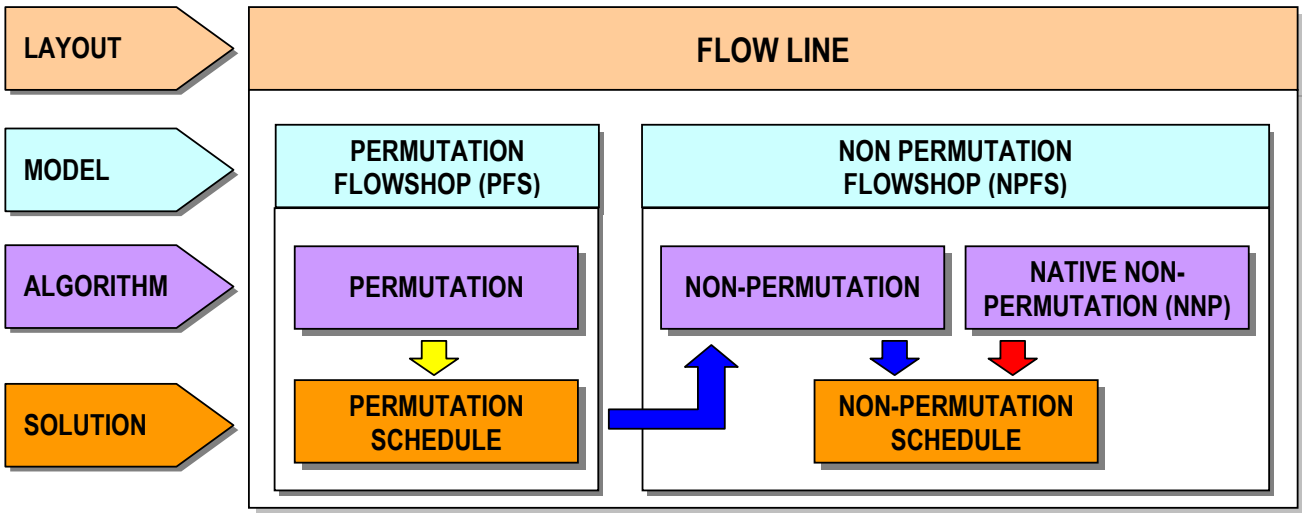

Figure 1 Solution generation according to permutation, non-permutation and native non-permutation (NNP) approaches.

Figure 1 about here

The logical steps, from a physical layout to a model, and the algorithm selection options are outlined in Figure 1.

In job passing, a job can overtake another job while waiting in a queue to be processed by a machine. To allow job passing, buffers between machines are necessary. Buffers can be between,

on board or shared among machines; they can be shared in the form of an automatic warehouse or an open space.

The optimal solution of the flow shop scheduling problem can be determined in polynomial time when $m=2$ (Johnson, 1954), the general form of this kind of problem is known to be NP-complete in the strong sense when $m \geq 3$ (Garey *et al.*, 1976).

Permutation flowshop has been extensively investigated in the literature, as opposed to non-permutation. A possible reason is that there are $(n!)^m$ different schedules for ordering jobs on machines in non-permutation flowshop; the number of schedules for permutation flowshop (PFS) reduces to $n!$

In continuous lines, with bulky or difficult to handle items, and when buffers are not present, job passing between machines is not allowed. In this case, modeling a flow line as a permutation flowshop is the only choice.

In other cases, modeling a manufacturing system as a non-permutation flowshop has several benefits. Among drawbacks of permutation flowshop are:

1. If buffers are not present, either the *blocking* or the *no-wait* condition should be applied to the algorithm to achieve a feasible scheduling. In the former case, a job completed on one machine may block that machine until the next downstream machine is free, while in the latter the next machine must be available before a job leaves the previous one.
2. Different permutations of jobs can be selected for subsequent machines, removing the constraint that all the sequences must be identical. By relaxing the permutation constraint, non-permutation schedules are potentially better. Experimental evidence is also available in the literature (Liao *et al.*, 2006), however due to the problem complexity, higher computation power is necessary.
3. Heuristics approaches are easier to implement and provide good permutation schedules for practical purposes with lower computation time and are therefore preferred by the shop-floor manager, but unfortunately, this simplicity is bought at the price of drastically inferior schedules according to Potts *et al.* (1991), and Tandon *et al.* (1991). More recently, Pugazhendhi *et al.* (2002), Liao *et al.* (2006), and Ying *et al.* (2010) have overcome the permutation performance proposing better results (lower upper bounds) for benchmarks in non-permutation configuration; they are currently the state-of-the-art.
4. In the application of permutation flowshop scheduling to a hybrid (flexible) flowshop system, to keep the same permutation on all stages it is required that jobs sorted by processing time have the same sorting on the different stages. Examples are the SPT (shortest processing time) and the LPT (longest processing time) dispatching rules. If jobs do not have the same sorting by

processing time, the resulting permutation scheduling is not optimal, because permutation scheduling requires that jobs are processed in the same sequence. An example follows (Figure 3).

N. of identical machines: 2 at stage *j*: 2 at stage *j*+1

N. of identical machines: 2 at stage *j*+1: 3 at stage j+1

Processing time at stage *j*: T for J1; 3T for J2; T for J3; T for J4

Processing time at stage *j*+1: T for J1; T for J2; T for J3; T for J4

PFS sequence (on both stages): J1, J2, J3, J4

Makespan for PFS: 5T

Optimum sequence (for NPFS)

on stage *j*: J1, J2, J3, J4

on stage *j*+1: J1, J3, J2, J4

Makespan for NPFS: 4T

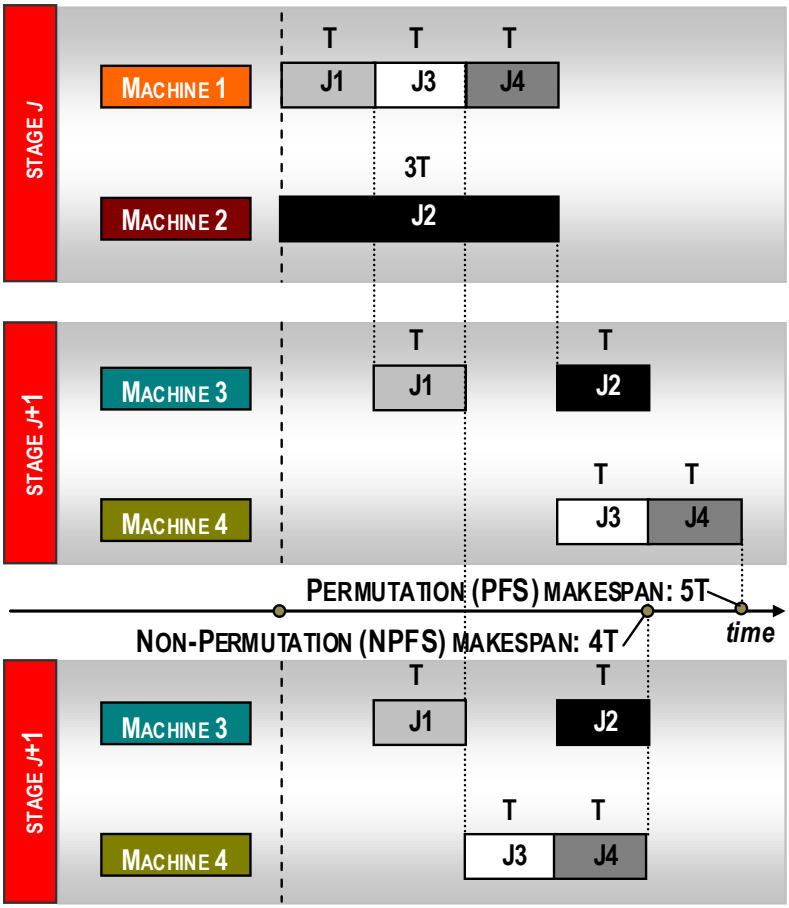

Figure 2 Example of reduced makespan by non-permutation flowshop scheduling in hybrid (or flexible) flowshop.

Figure 3 about here

5. In sequence-dependent setup, permutation scheduling is not optimal, because different sequences of jobs on different machines may share the same setup or incur lower setup change penalties. Instead, permutation flowshop scheduling imposes the same sequence on all machines. An example follows.

   PFS sequence: J1, J2, J3

   Setup type at stage *j*: A for J1 and J3; B for J2

   Setup sequence for PFS: A, B, A

   Setup sequence for NPFS: A, B

To deal with the complexity of the non-permutation flowshop problem, a number of approaches are available, which improve an optimal permutation schedule by changing the job sequences on machines, as shown by the arrow paths in Figure 1. Examples are the well-known NEH heuristic to find good permutation solutions, further improved by metaheuristics, like ant colony optimization (ACO), genetic algorithms (GA), Particle Swarm Optimization (PSO, Vijay chakaravarthy *et al.*, 2011), and tabu search, to achieve non-permutation solutions. Lin and Ying (2009) propose a hybrid simulated annealing/tabu search method where job sequences are the same for all machines in the initial solution of their tabu search method. Pughazendi *et al.* (2004) propose a simple heuristic procedure to derive non-permutation schedules from a given permutation sequence, with the added complexity of sequence-dependent setups. Brucker *et al.* (2003) present a number of results for non-permutation flowshop scheduling, with limited capacity buffer. They propose a procedure based on state-of-the-art tabu search, where the initial solution adopted is a permutation schedule evaluated by the NEH heuristic (Nawaz *et al.*, 1983). Jain and Meeran (2002) consider a multi-level hybrid approach for the general flowshop scheduling problem. They hybridize a scatter search and a tabu search with the neighborhood structure proposed by Nowicki and Smutnicki (1996) for the job-shop scheduling. The initial solution is found by the insertion algorithm proposed by Werner and Winkler (1995) and it is implemented on the disjunctive graph. In general, the tabu search requires starting from a good initial solution to be improved by other heuristics.

ACO has been recently considered to cope with the complexity of other flowshop scheduling systems (Arnaout *et al.*, 2012). According to the recent literature review from Tavares Neto and Godinho Filho (2013) there are six most used ACO algorithms applied to scheduling problems, which were proposed between 1991 and 2000. Among approaches to flowshop scheduling, Stuezle (1998) proposed a Min-Max Ant System from Stuetzle and Hoos (2000) for the permutation flowshop scheduling. The proposed Ant Colony System (ACS) is from Dorigo and Gambardella (1997) as for Rajendran and Ziegler (2004). Similarly, Yagmahan and Yenisey (2010) consider a multi-objective makespan and flowtime criteria. They also propose to create initial ant trails with an

amount of pheromone that is a function of the best solution generated by the NEH heuristic by Nawaz *et al.* (1983). This idea has been exploited by Sadjadi *et al.* (2008) who applied the standard ACO specifications from Bonabeau *et al.* (2000) except for the diversification mechanism: the initial amount of pheromone on the digraph trails is determined by the permutation solution found by the NEH heuristic. The best-found permutation schedules are improved by *local search* in order to obtain non-permutation solutions.

To the best of our knowledge, a native ACO has been applied only by Ying and Lin (2007). They represent the problem by a disjunctive graph and use a multi-heuristic function of *visibility* in an ant colony system. The visibility represents the heuristic desirability, that together with the pheromone amount, drives an ant to the selection (and to the direction) of disjunctive arcs of the digraph. The multi-heuristic visibility proposed by Ying and Lin is based on the best selection within a set of schedules achieved by dispatching rules. The performance is evaluated using the benchmark problems from Demirkol *et al.* (1998), achieving new best results (lower upper bounds) on most benchmarks considered. Their ACO improves the previous best results obtained by the shifting bottleneck heuristic by Demirkol *et al.* and are compared with the proposed NNP-ACO. Extensive numerical research has indicated that the shifting bottleneck heuristic is extremely effective (Pinedo, 1995).

2 The native non-permutation approach

As opposed to all cited works, native metaheuristic approach enforces constructive schedules. Initially, it builds independent (arbitrary) initialization sequences on all machines. Next, a schedule is generated ex-novo by each ant.

The native approach prevents the algorithm performance to be influenced by other (meta)heuristics and allows an accurate performance evaluation.

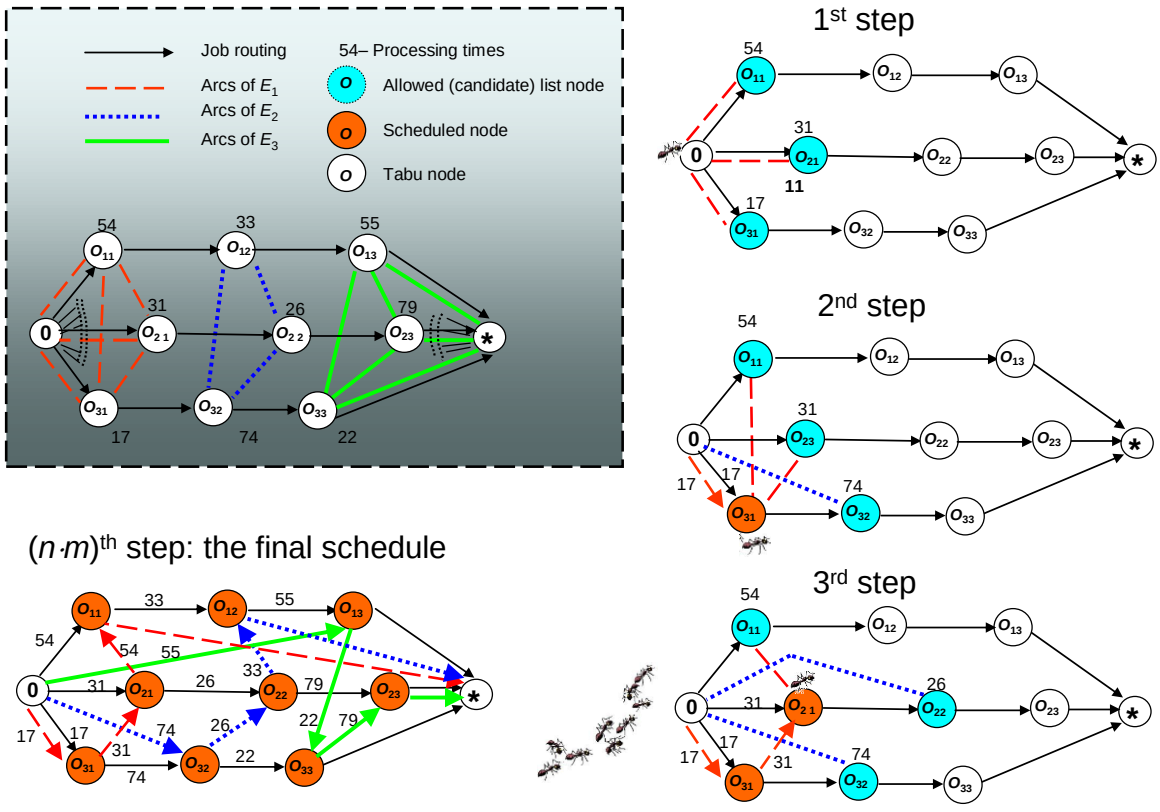

Figure 3 Digraph approach for candidate list selection (inset) and building a feasible solution.

Figure 3 about here

A possible representation of the (non-permutation flowshop) scheduling problem is based on a disjunctive graph. In a disjunctive graph (Figure 3, inset), nodes represent operations, conjunctive arcs correspond to precedence constraints among the operations for the same job, disjunctive arcs conform to possible constraints among the jobs on the same machine.

We define as *native non-permutation flowshop scheduling* (NNPFS) an algorithm able to generate constructive solutions in O($n \cdot m$). A native non-permutation flowshop scheduling algorithm directs some disjunctive arcs in order to connect all operations with only a starting and an ending arc, i.e. the graph becomes *acyclic*, and the related sequences of operations on machines represent a *feasible schedule*. The makespan C_{max} is the length of the longest path between the dummy source and destination (*critical path*). When a disjunctive arc is directed, the ending node is inserted in the sequence of the related machine and in a tabu list, i.e. the partial schedule generated on the digraph cannot be modified anymore. Consequently, the longest path between the dummy source and the ending node is the *completion time* of the operation. Generally, the constructive mechanism is implemented by the *list-scheduling* algorithm (Giffler and Thompson, 1960). Besides, initially it

starts with a random selection of nodes or a type of selection driven by dispatching rules. These features make list-scheduling mandatory in native non-permutation flowshop scheduling.

A typical digraph framework metaheuristics is the ant colony optimization proposed by Bonabeau *et al.* (2000). In ACO, the digraph is an internal shared memory where, in analogy with nature, artificial ants, following trails of pheromone, direct disjunctive arcs to connect source and destination nodes (Figure 3). Each ant of the colony directs a number of disjunctive arcs in order to make the graph acyclic; finally, the ant that connects source and destination nodes with the shortest critical path (Figure 3, bottom left) is eligible to leave a pheromone amount proportional to the path length. This is a constructive way to generate a schedule. A complete solution is generated forward by a partial solution using the stigmergy of the colony, i.e. the selection of the more promising disjunctive arcs where a higher amount of pheromone is laid. The main goal of the ACO mechanism is to generate optimal solutions by constructive schedules. The concept is similar to "divide et impera", because the stigmergy progressively concentrates the search in a low number of very small promising regions. Differently to local search, this fact makes the algorithm intrinsically parallel.

3 Problem formulation

Current problem is referred as $F_m|B_i=+\infty|C_{\max}$ using Graham's notation, where

F_m stands for flowshop with m machines, only one machine is present at each stage,

$B_i=+\infty$ denotes that buffers with infinite capacity are present, allowing non-permutation schedules, and

$C_{\max}$ denotes the makespan minimization as the optimization criterion.

The n jobs have to be scheduled on the m machines. Each job i, i=1,..,n, is represented by a sequence of operations, $O_{i\,j}$; each job has to be processed as the j^{th} operation on the machine at stage j, j=1,..,m, with a processing time $t(O_{i\,j})$; a machine j coincides with the belonging stage.

Given a schedule S, the quantities $st(O_{i\,j})$ and $L(O_{i\,j})$ denote, respectively, the starting and the completion time of the operation $O_{i\,j}$.

The following conditions apply:

- no resource can process more than one operation at a time;
- no operation $O_{i\,j}$ can start until $O_{i\,(j-1),}$ is completed;
- no operation can stop after its start (no *preemption* allowed).

The problem (Figure 3, inset) can be represented by a disjunctive graph $DG = (N, A, E_j, W_N)$ where

N is the set of nodes (operations); the set of nodes N has $(n \cdot m)$ elements plus two dummy nodes: the source "0" and the destination "*";

A is the set of conjunctive arcs (job routing); the set of conjunctive arcs A includes $[(n+1) \cdot m)]$ elements;

E_j is the set of disjunctive arcs (connecting a potential pair of operations to include in the sequence of machine/stage j); E_j includes $[(n-1) \cdot n/2]$ disjunctive arcs, among all the n operations to be processed on machine j, and $(2 \cdot n)$ between each operation to be processed on machine j and the dummy nodes;

W_N is the weight on nodes (processing times, setup and transportation times etc.).

A feasible solution can be constructively obtained by list-scheduling. This heuristic generates a feasible solution in $(n \cdot m)$ steps by inserting an *allowed* operation in the related machine sequence at each step. An operation is included in an *allowed list* if it can be reached by a conjunctive arc of A from the previous operation in the job routing that is already inserted in the machine sequence. When an operation is selected by list-scheduling, it is removed from the allowed list and inserted in a tabu list.

Figure 3 shows an example of how the solution can be achieved on the digraph and will be detailed in the next chapter showing the implementation of the proposed ACO. Initially, the dummy source "0" is the starting node. At the first step, all the operations that can be reached by a directed arc of A are the operations that must be processed by machine 1: O_{11}, O_{21} and O_{31} (shown in light blue). At the second step, operation O_{31} is selected and inserted in the sequence of machine 1 (shown in red). At the second step, the disjunctive arc $(0, O_{31}) \in E_1$ is directed. Consequently, O_{31} is replaced from the allowed list by the next operation reached from O_{31} by an arc of A, i.e. O_{32}. The length 17 in bold represents the completion time of operation O_{31} in the partial constructive schedule, i.e. $L(O_{31})=st(O_{31})+t(O_{31})=17$. At the third step, the disjunctive arc $(0, O_{21}) \in E_1$ is directed and O_{21} is replaced from the allowed list by the next operation reached from O_{21} by an arc of A, i.e. O_{22}. The completion time $L(O_{21})=31$ is evaluated as the longest path from the dummy source "0". This path length is evaluated by adding the processing time $t(O_{21})=31$ to the maximum length between the two nodes from which the current node can be reached, respectively: O_{31} from the just directed arc, and the dummy "0" from the related routing arc of A. It can be noticed that the maximum completion time between O_{31} and "0" is the starting time of node O_{32}, $st(O_{32})$. The algorithm runs for $[(m \cdot n)-2]$ more steps generating the acyclic graph of Figure 3: at the $(m \cdot n)^{th}$ step all nodes will turn red and their completion time will be displayed. The makespan is evaluated from the node with the maximum path length.

4 The proposed ACS

This section describes an ant colony system (ACS) for the native non-permutation flowshop scheduling problem. Ant colony systems, a subset class of ACO, are an emerging class of biologically inspired research dealing with artificial or swarm intelligence, which exploits the experience of an ant colony as a model of self-organization in co-operative food retrieval. Ants run the nest-food path by a probabilistic selection of nodes according to the following properties:

i) *diversification* in order to produce promising alternative paths;

ii) *intensification* to select a node in the vicinity of the current best paths.

As soon as all the paths of the ants in the colony are generated, the best ant deposits on the arc a further amount of pheromone proportional to the path length and a pheromone *decay* routine is performed to prevent the stagnation in local optima solutions. The *pheromone trail* is the basic mechanism of communication among real ants and it is mimicked by the ant colony system in order to find the shortest path, connecting source and destination on a weighted graph, which represents the optimization problem.

The two inverse mechanisms are achieved by negative and positive pheromone deposition, respectively through the local update rule and off-line pheromone update rule. Diversification by the local update rule pushes towards permutated schedules and is the core mechanism to generate natively non-permutation solutions.

List-scheduling, which is a process guided only by the function of visibility, becomes driven also by the pheromone amount. In fact, the selection of a candidate node can be performed based on the disjunctive arc that connects it. At the start, the pheromone is randomly deposited; consequently, the node selection is random as in pure list-scheduling algorithms. As epochs evolve, the deposited pheromone drives the arc selection.

The following mechanisms have been implemented in the proposed ACO and are described in detail:

- path generation, to transform the digraph in an acyclic conjunctive graph by a stochastic process based on the amount of pheromone;
- candidate list construction, to select operations, not only to achieve feasible schedules, but also in order not to exceed the idle time more than a predefined value;
- local update rule, for diversification purposes;
- off-line pheromone update rule, for intensification purposes;
- local search, to better improve the best found solution.

In addition to fulfill the requirements of a native non-permutation approach, the following innovative mechanisms have been implemented in the proposed ACO: list scheduling, and freezing and stability conditions.

4.1. Digraph model

The disjunctive graph is also able to implement pheromone trails. In this case the $DG = (N, A, E_j, W_N, W_E)$ has the new component W_E, which represents the weight on disjunctive arcs (pheromone amount). The weight on the disjunctive arcs $(O_{i'j}, O_{ij})$ of E_j is represented by the pair $W_E(O_{i'j}, O_{ij}) = (\tau[O_{i'j}, O_{ij}], \tau[O_{ij}, O_{i'j}])$. The first component array $\tau[O_{i'j}, O_{ij}]$ gives an index of desirability in order to insert the feasible move (i.e. the sub-sequence) $[O_{i'j}, O_{ij}]$ in the current location of the loading sequence of resource j. The pheromone trail W_E is a two-dimensional array of $[m \cdot n \cdot (n+1)]$ elements (considering that the number of disjunctive arcs among all the n operations to be processed on machine j that can be replicated is $[n\ (n-1)/2]$). Hence, the internal shared memory of the proposed ant colony system is $O(m \cdot n^2)$.

4.2. Path generation

To generate a feasible schedule S_a, each ant a visits every operation on DG one and only one time in order to transform the digraph in an acyclic conjunctive graph. Path generation is a stochastic process where an ant starts from the dummy "0" and selects the next node from a subset of the allowed operations, the *candidate list CL*. It uses the following *transition probability* rule of the pheromone trail as a function of both the heuristic function of desirability, η (the visibility function), and the amount of pheromone τ on the edge (O_{ij}, J), with $J \in CL$:

$$Z = \begin{cases} \arg\min_{O_{ij} \in CL} \left\{ \tau(O_{ij}, J)^{\alpha} \cdot \eta(O_{ij}, J)^{\beta} \right\}, & \text{if } q \le q_0 \\ J, & \text{if } q > q_0 \end{cases} \quad (1)$$

The non-negative parameters α and β represent respectively the intensity of the amount of pheromone and the visibility included in the transition probability function. The non-negative parameter q_0 is the *cutting exploration*, a mechanism that restricts the selection of the next operation from the candidate list *CL*. If a random number q is higher than the cutting exploration parameter q_0 ($0 \le q_0 \le 1$), the candidate operation is selected examining in probability all the candidate operations

that are as much desirable as higher visibility and pheromone amount are; otherwise the most desirable operation is taken, i.e. the arc with the highest amount of pheromone and the highest visibility.

The role of cutting exploration is that of explicitly split the search space in order to achieve a compromise between the probabilistic mechanism adopted for $q \leq q_0$ or the further intensification mechanism of exploring near the best path so far, which corresponds to an exploitation of the knowledge available about the problem. By tuning parameter q_0 near 1, cutting exploration allows the activity of the system to concentrate on the best solutions (*exploitation activity*) instead of letting it explore constantly (*exploration activity*, achieved by tuning parameter q_0 near 0). In fact, when q_0 is close to 0, all the candidate solutions are examined in probability, whereas when q_0 is close to 1, only the local optimal solution is selected by equation. In this paper a *freezing function* similar to one proposed by Kumar *et al.* (2003) is considered. It progressively freezes the system by tuning q_0 from 0 to 1, in order to favor the exploration in the initial part of the algorithm by means of the following expression:

$$q_0 = \frac{\ln(epoch)}{\ln(n\_epochs)} \tag{2}$$

where *epoch* is the current iteration and *n_epochs* is the total number of iterations of the ant colony system.

The heuristic function of desirability η is a very critical component of ant colony systems. Generally, it is implemented by dispatching rules, as in Ying and Lin (2007). A comparison among a number of dispatching rules to implement the visibility function has been performed by Blum and Sampels (2004). In this paper the earliest stating time (EST) heuristic, i.e. the best heuristic tested by Blum and Sampels (2004), is used.

4.3. Candidate list construction

The candidate list does not include all the operations that can be selected at a given construction step of the algorithm. In fact, 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: in these schedules no machine is kept idle when it could start processing some operations.

As no 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 machine is kept idle for more than a predefined value δ if it could start processing some operations. As the minimum starting time of a candidate operation is:

$$\min_{O_{ij} \in AL} \; st(O_{ij}) \tag{3}$$

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

$$st(O^*) \leq \min_{O_{ij} \in AL} \; st(O_{ij}) + \delta, \qquad O^* \in AL \tag{4}$$

In this paper a strategy which relaxes the expression (4) is considered by using the following parametric value of δ:

$$\delta(rf) = \frac{\max_{O_{ij} \in AL} st(O_{ij}) - \min_{O_{ij} \in AL} st(O_{ij})}{rf} \tag{5}$$

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

$$CL = \left\{ O^* \in AL \;\middle|\; st(O^*) \leq \min_{O_{ij} \in AL} st(O_{ij}) + \frac{\max_{O_{ij} \in AL} st(O_{ij}) - \min_{O_j \in AL} st(O_{ij})}{rf} \right\} \tag{6}$$

where a restricted factor *rf* equal to 3 is considered.

4.4. Local update rule

The local update rule is applied to favor the exploration of not visited nodes by other ants of the colony. This rule imposes to the ant that has selected a candidate operation *J*, of laying on the connecting arc ($O_{i'j}$, *J*) the following negative amount of pheromone:

$$\tau(O_{i'j}, J) = (1-\rho) \cdot \tau(O_{i'j}, J) + \rho \cdot \tau_0 \quad (7)$$

This mechanism enhances diversification, a fundamental mechanism to generate permutated schedules (native non-permutation approach). The local update rule is a convex combination of the evaporation coefficient parameter; the convex combination has points, $\tau(O_{i\,j}, J)$ and τ_0. This causes a reduction of the pheromone amount on a selected arc, because it ranges between the previous value $\tau(O_{i\,j}, J)$ and the initial value τ_0. The effect of this rule is making nodes less and less attractive as they are visited by ants, indirectly favoring the exploration of not visited nodes. This is a basic diversification mechanism because it makes the generation of alternative paths by next ants possible.

4.5. Off-line pheromone update rule

This feature arises when a positive amount of pheromone has to be deposited. At the end of an epoch, in order to make a more directed exploration of the best nest-food paths by the entire colony, the ant which detects the best path in the epoch, termed best-epoch ant (S_{be}) performs an *off-line update rule* of pheromone. This rule consists in depositing on the acyclic graph generated by S_{be} a further amount of pheromone, proportional to the following convex combinations of points $\tau(O_{i\,j}, J)$ and $makespan(S_{be})^{-1}$, making a search intensification, by other ants of the colony, in the vicinity of the best solution possible:

$$\tau(O_{i\,j}, J) = (1-\rho) \cdot \tau(O_{i\,j}, J) + \rho \cdot makespan(S_{be})^{-1}, \qquad (O_{i\,j}, J) \in S_{be} \quad (8)$$

$$= (1-\rho) \cdot \tau(O_{i\,j}, J), \qquad \text{otherwise}$$

The amount of pheromone remaining on the selected path ranges between the previous value, $\tau(O_{i\,j}, J)$, and closer value to the optimum: $makespan(S_{be})^{-1}$. On the contrary, a routine of pheromone decay on the pheromone trails is performed on the other arcs of the digraph, indicating that a rarely used path probably does not lead to optimal solutions.

4.6. Local search

When a path is generated, the solution is lead to its local optimum by a *local search* routine. A steepest descent algorithm is considered, where, at each iteration, current best solution is replaced with an improved solution in its neighborhood. The local search performance depends on the neighborhood structure. Here, a neighborhood structure derived from the state-of-the-art local search for job-shop scheduling, proposed by Nowicki and Smutnicki (1996) for their fast tabu search algorithm, is adopted.

4.7. Stability condition

The stop criterion is sometimes a fixed epoch number or computation time. Instead, a stability condition stops the algorithm: a fixed number of epochs (3000) producing an error reduction of at least one processing time unit per epoch.

4.8. ACS implementation

The following pseudo-code gives a high-level description of an Ant Colony System for Native Non-Permutation Flowshop Scheduling.

Input: a weighted digraph $WDG=(N, A, E_j, W_N, W_E)$

// **Initialization**

for each disjunctive arc $(O_{i'j},O_{ij})$ of E_A deposit a small constant amount of pheromone $W_E(O_{i'j},O_{ij})$ $= (\tau_0, \tau_0)$ where $\tau_0 = \left[n \cdot m \cdot \max_{j=1,..,m} \sum_{i=1}^{n} t(O_{ij}) \right]^{-1}$

$epoch \leftarrow 1$; $not\_improve \leftarrow 0$;

// Main Loop

while (*not_improve* < *stability_condition*) **do**

// Epoch Loop

for each ant *a*, *a* = 1 **to** *colony_size* **do**

// Path Generation

$S_a \leftarrow \varnothing$;

1. $O \leftarrow \{O_{ij} \mid i=1,..,n, j=1,..,m\}$;
2. *Inizialization of Candidate Nodes:* $AL_1 \leftarrow \{O_{ij} \mid i=1,..,n, j=1\}$;

for each w =1 **to** $n \cdot m$ **do**

3. *Restriction: restrict* AL_w to the candidate list CL_w by means of expression (6);
4. *Inizialization of Feasible Moves* (i.e. the disjunctive arcs connected to operation of CL_w*);*
5. *Move Selection*: select a feasible move ($O_{i'j}$, $O_{i\ j}$) of E_j, where $O_{i\ j} \in CL$, by means of the transition probability rules (1); directing the related disjunctive arc ($O_{i'j}$ = dummy "0", if j =1);
6. *Arc Removing:* remove all disjunctive arcs connected to $O_{i'j}$ (i.e. no other operation can be immediately subsequent to $O_{i'j}$ in the machine sequence);
7. *Path length evaluation:* the length $L(O_{i\ j})$ of the longest path connecting $O_{i\ j}$ to the dummy "0" is evaluated by $L(O_{i\ j}) = t(O_{i\ j}) + \max\{L(O_{i'j}), L(O_{(i-1)\ j})\}$ and it is placed as a mark near the scheduled operation;
8. *Local Updating*: apply local update rule (8) to arcs ($O_{i'j}$, $O_{i\ j}$) $\in W_E$;
9. *Update Allowed List:* remove the scheduled operation from the allowed list,

   $AL_w \leftarrow AL_w \cup \{O_{i\ (j+1)}\} / \{O_{i\ j}\}$, if $j \leq m$-1;

   $\leftarrow AL_w / \{O_{i\ j}\}$, otherwise;

end for

10. Directing the remaining disjunctive arcs (these arcs are connected to dummy "*")
11. *Local Search*: Apply local search with neighborhood structure of Nowicki and Smutnicki (1996) to S_a;
12. *Best Evaluation:* **if** (*makespan*(S_a) < *makespan*(S_{be}))

    then (*makespan*(S_{be}) ← *makespan*(S_a) **and** $S_{be} \leftarrow S_a$)

    end if

end for

Global Updating: Apply global update rule (5);

Best Ant Evaluation: **if** (*makespan*(S_{be}) < *makespan*(S^*))

then ((*makespan*(*S**) ← *makespan*(S_{be}); *S** ← S_{be} **and** *epoch* ← 0)

and

not_improve ← 0;

else *epoch* ++ **and** *not_improve* ++;

end if

end while

Output: *S**

5 Computation experiments

The proposed heuristics have been verified by computation experiments on well-known benchmark problems for comparison with other approaches using an identical test set. Selected benchmarks are from Demirkol *et al.* (1998), are enclosed as supplementary material and downloadable from http://tinyurl.com/demirkol (or http://www.ing.unipi.it/lanzetta/demirkol/dataset.txt). Benchmarks are arrays $b_{n\times m}$. The $n\times m$ operations of each job on all machines are represented by their processing times. Each benchmark instance k is characterized by a lower bound ($LB_{n\,m\,k}$). The lower bound is the minimum theoretical target for the objective function, in this case the makespan $C_{\max}$. If the makespan found by an algorithm coincides with the lower bound, the global optimum has been reached. The best-known makespan is assumed as the benchmark upper bound (UB).

Metrics for algorithm performance are the *%gap* from the lower bound of benchmark instances $b_{n\times m}$ of the original dataset and from the best-known solution of other researchers, e.g.:

$$\%gap_{m\,n\,k} = \frac{C_{best_{m\,n\,k}} - LB_{m\,n\,k}}{LB_{m\,n\,k}} \tag{9}$$

The proposed Native Non Permutation Ant Colony Optimization (NNP-ACS) is compared with the Shifting Bottleneck DMU-SB from Demirkol *et al.* (1998) and the Multiheuristic Desirability Ant Colony System MHD-ACS from Ying and Lin (2007). These three algorithms are visually compared in Figure 4 and detailed in Table 3.

The benchmark set adopted herein was established by Demirkol, Mehta, and Uzsoy (DMU) and includes 40 test instances, generated as follows. All jobs are available at time zero, and the operation processing times are generated by a discrete uniform distribution between 1 and 200.

Four job number values n=20, 30, 40 and 50 and two machine number values m=15 and 20 result in eight combinations of m and n and a total number of operations on all jobs and machines ranging from 300 to 1000.

The ratio of n to m varies between 1 and 3.3. These combinations yield a problem set that is not based on a specific application. Ten instances were generated for each of the eight combinations of m and n. Since the size and complexity of the instances make exact solutions impractical, Demirkol *et al.* solved each instance by five different constructive heuristics and three versions of the shifting bottleneck procedure. All algorithms were run on a SUN SPARC® server 1000 Model 1104 with a 50 MHz processor, which is a multitasking system running under Unix®. The upper bound for each instance was the best solution found by any of the algorithms.

The best available solutions for the test problem set in the literature appear to be those proposed by Demirkol *et al.* (1998). A lower bound (LB) was obtained for each instance by relaxing the capacity constraints on all but one machine and solving to optimality the resulting single machine problem of minimizing makespan with release and delivery times (Demirkol *et al.*, 1998). Demirkol *et al.* performed this method for each machine, and reported the highest makespan value obtained as a lower bound in each instance. To obtain a more compact and challenging set of test problems, Demirkol *et al.* ranked the instances according to decreasing order of %gap between the upper and lower bounds for each combination of m and n. Only the first five instances for each combination were finally presented. Thus, a total of 40 test instances were obtained as listed in the first three columns of Table 3 and denoted by the term *flcmax_n_m_k*.

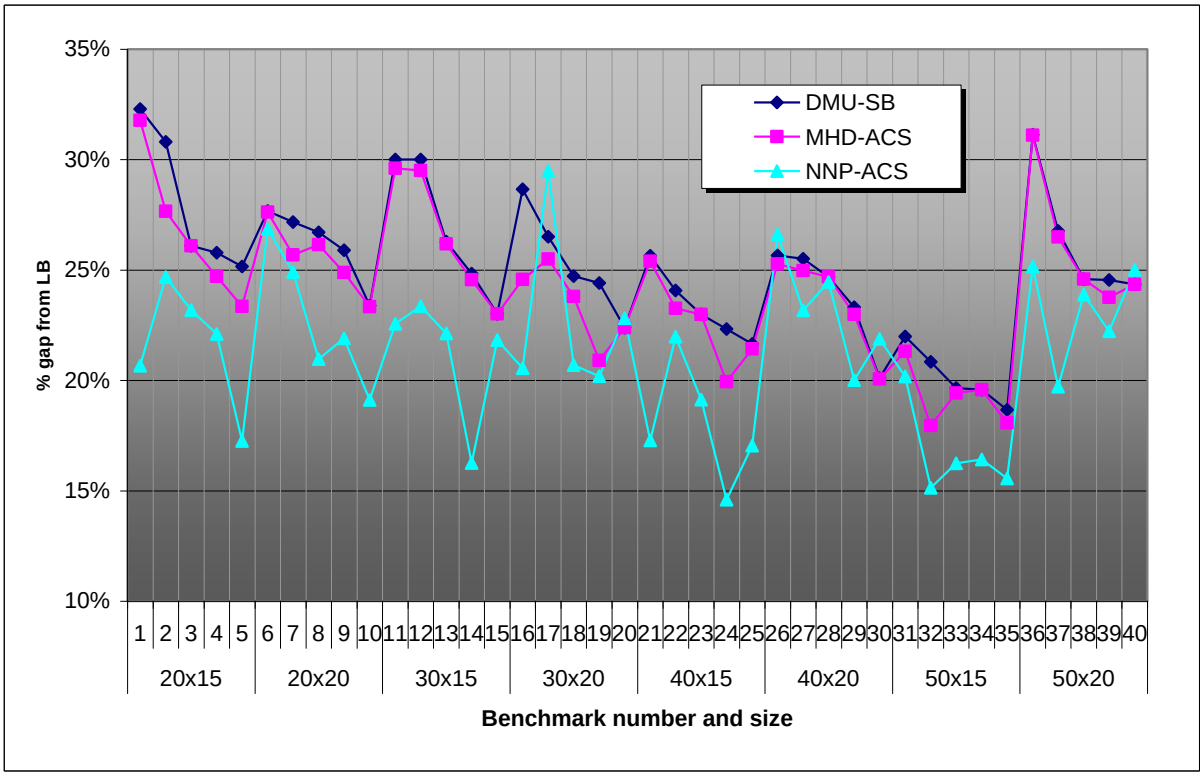

Figure 4 *%gap* (9) of tested algorithms on benchmark problems.

Figure 4 about here

The *%gap* defined in (9) for the original dataset (DMU-SB) is graphically shown by the blue diamonds in Figure 4 and ranges between 32 and 18%. The periodic behavior of error curves is observed in each subset of 5 benchmark instances of increasing size (increasing machine number and/or job number), and to the decreasing difficulty according to which benchmark instances have been originally sorted. Comparing the trends among subsets of 5 benchmark instances of size $n \times m$, some differences can be noticed: both for n=20 and n=30 the *%gap* slightly decreases from m=15 to 20 (from 32-25% to 28-23% and from 30-23% to 29-22%). For n=40 the trend of the *%gap* within the subsets for m=15 and for m=20 is similar (26-22% and 26-20%). For n=50 the *%gap*, inversely with respect to smaller benchmarks, the *%gap* increases from 15 to 20 machines (from 22-19% to 31-24%). This visual analysis shows the different intrinsic difficulty of benchmarks.

The best makespan of 5 runs of the MHD-ACS algorithm and its corresponding time are listed for comparison within the same class of native non-permutation algorithms as the proposed NNP-ACS in Table 3.

Preliminary tests have been run with the main parameters in Table 1. The tested parameters from the previous ACO approach are also listed.

Table 1 Tested parameters for the ant colony systems under comparison from Ying and Lin (2007) and from the proposed NNP-ACS.

Parameter	MHD-ACS	NNP-ACS
colony_size	5, 10, 20, 50, 100	5, 6, 7, 8
α	$(0.1 + 0.2 \cdot i)$, i=1,…,5	0.1, 0.2, 0.5, 1, 1.5, 2
β	0.1, 0.5, 1, 2, 5	$(0.04 \cdot i)$, i=1,…,9
q_0	$(0.1 + 0.2 \cdot i)$, i=1,…,5	$\frac{\ln(not\_improve + 1)}{\ln(stability\_condition)}$
ρ	$(0.1 + 0.2 \cdot i)$, i=1,…,5	$(0.1 \cdot i)$, i=1,…,8
Visibility function	MHD	EST, PAST (Rossi and Dini, 2007)

The selected parameters used in the comparison on benchmarks are in Table 2. The proposed NNP-ACS has run 10 times on 3 GHz 32 bit Intel® Pentium® IV based PCs with 2 GB RAM on each of the 40 benchmark problems, providing a total of 400 results. The average and the best solutions for each of the 40 instances along with its corresponding time are reported in Table 3. A scale factor of two, based on the speed of PC processor used by respective authors, can be applied to the processing time for comparison with the MHD-ACS algorithm.

Table 2 Selected non-optimal parameters for the ant colony system in Ying and Lin (2007) and the proposed NNP-ACS.

Parameter	MHD-ACS	NNP-ACS
Colony size	5	8
τ_0	$\left[n \cdot m \cdot UB\right]^{-1}$	$\left[n \cdot m \cdot \max_{j=1,..,m} \sum_{i=1}^{n} t(O_{ij}) \right]^{-1}$
α	0.1	2
β	2	0.3
Epoch number	5000	*not_improve* < *stability_condition*
Stability_condition	NA	3000
q_0	0.9	$\frac{\ln(not\_improve+1)}{\ln(stability\_condition)}$
ρ	0.1	0.12
Local search	NA	steepest descent (Nowicki and Smutnicki, 1996)
Visibility function	MHD	EST
Candidate list	NA	$\left\{ O^* \in AL \mid st(O^*) \le \min_{O_{ij} \in AL_w} st(O_{ij}) + \frac{\max_{O_{ij} \in AL_w} st(O_{ij}) - \min_{O_j \in AL_w} st(O_{ij})}{rf} \right\}$

5.1. Results

Table 3 lists the results of the state-of-the-art ant colony system algorithm from Ying and Lin (2007) and those achieved with the described NNP ant colony system. Figure 4 visually displays the performance of the three native non-permutation algorithms under comparison with respect to the LB using the metrics in (9).

In **bold** the new upper bounds for the three native approaches. For 35 of the 40 benchmarks examined, new upper bounds have been found. The only original upper bounds from Demirkol *et al.*, which have not been met, are flcmax_30_20_2 and flcmax_50_20_4.

Table 3 Benchmark problems, state-of-the-art solutions, and results of computation experiments.

Algorithm →	Bench-mark	DMU-SB	MHD-ACS		NNP-ACS		
Instance ↓	LB	C_{best}	C_{best}	Time [s]	C_{best}	Time [s]	C_{avg}
flcmax_20_15_3	3354	4437	4420	46	**4047**	330	4113.4
flcmax_20_15_6	3168	4144	4044	46	**3950**	36	3977.5
flcmax_20_15_4	2997	3779	3786	46	**3692**	114	3730.6
flcmax_20_15_10	3420	4302	4265	45	**4176**	96	4221.7
flcmax_20_15_5	3494	4373	4310	45	**4097**	66	4124.9
flcmax_20_20_1	3776	4821	4819	59	**4790**	30	4826.9
flcmax_20_20_3	3758	4779	4723	60	**4694**	114	4715.7
flcmax_20_20_9	3902	4944	4922	60	**4720**	282	4775.4
flcmax_20_20_2	3881	4886	4847	60	**4731**	282	4781.3
flcmax_20_20_10	3823	4717	4715	60	**4554**	24	4607.6
flcmax_30_15_3	4020	5226	5210	93	**4927**	438	5032.2
flcmax_30_15_4	4080	5304	5284	94	**5033**	78	5092.4
flcmax_30_15_9	4022	5079	5075	95	**4912**	114	4968.6
flcmax_30_15_8	4490	5605	5593	94	**5220**	42	5320.2
flcmax_30_15_6	4184	5147	5149	93	**5097**	456	5158.5
flcmax_30_20_3	4806	6183	5987	121	**5794**	84	5846.9
flcmax_30_20_1	4772	6037	**5989**	124	6179	126	6221.8
flcmax_30_20_6	5004	6241	6195	124	**6039**	84	6133.9
flcmax_30_20_10	4899	6095	5923	121	**5888**	54	5967.7
flcmax_30_20_2	4757	**5822**	5840	123	5842	798	5886.1
flcmax_40_15_5	5560	6986	6972	154	**6521**	180	6594.1
flcmax_40_15_9	5119	6351	6310	154	**6244**	354	6303.3
flcmax_40_15_2	5290	6506	6532	154	**6302**	78	6395.6
flcmax_40_15_10	5596	6845	6712	156	**6413**	1260	6445.2
flcmax_40_15_8	5576	6783	6771	156	**6526**	294	6611.7
flcmax_40_20_3	5693	7154	**7132**	210	7208	6	7274.5
flcmax_40_20_9	5998	7528	7496	208	**7388**	2262	7484.7
flcmax_40_20_6	5990	7469	7476	209	**7455**	936	7553.1
flcmax_40_20_7	6170	7608	7588	207	**7405**	1314	7473.5
flcmax_40_20_5	6011	7219	**7217**	210	7326	1614	7399.4
flcmax_50_15_6	6290	7673	7631	238	**7559**	120	7606.8
flcmax_50_15_5	6355	7679	7496	240	**7317**	2916	7368.4
flcmax_50_15_1	6198	7416	7402	240	**7205**	1266	7303.8
flcmax_50_15_8	6312	7548	7558	237	**7348**	318	7468.7
flcmax_50_15_2	6531	7750	7712	236	**7547**	762	7644.8
flcmax_50_20_2	6740	8838	8836	312	**8436**	738	8684.4
flcmax_50_20_1	6736	8539	8521	312	**8064**	2442	8189.7
flcmax_50_20_7	6756	8417	8425	313	**8370**	1470	8526

flcmax_50_20_8	6897	8590	8536	313	**8430**	6024	8509.2
flcmax_50_20_4	6830	**8493**	8502	312	8538	2472	8625.1

The range of the *%gap* of the proposed ACS is between 14.6% (flcmax_40_15_10) and 29.5% (flcmax_30_20_1) above the LB.

As also shown by the visual trend in Figure 4, performance are uniformly distributed in the benchmark size range from small to high.

The best makespan of the proposed algorithm is between 3.2% above (flcmax_30_20_1) and 8.4% below (flcmax_20_15_3) the best results from the competitor MHD approach. In 37 of 40 benchmarks, the best makespan found is lower. It has also been calculated that the average makespan of the proposed ACO (from the 10 runs for each benchmark) is lower than the best MHD solution on 31 out of 40 cases.

Similarly, comparing the *%gap* with the corresponding upper bound from Demirkol *et al.*, it ranges from 2.4% above (flcmax_30_20_1) to 8.8% below (flcmax_20_15_3). The best makespan found is lower in 38 out of 40 benchmarks.

On average, the *%gap* from the LB before current work was 24.22% and has been lowered now to 21.01%, still leaving room for further improvements and research.

6 Discussion

The features of the proposed ACS described in section 4 and implemented as in section 4.8 with the parameters in Table 1, have shown to improve the state of the art native non-permutation flowshop scheduling upper bounds. By watching the *%gap* pattern in Figure 4 it can be clearly observed that the proposed ACO is better performing than the competitors' approaches over the full range of available job and machine sizes.

ACO has been selected because of the following intrinsic features, which are able to produce natively non-permutation solutions to the flowshop problem:

- selection of nodes from the candidate list, which restricts the size of operations allowed, in addition to
- random initialization of pheromone and diversification mechanism by the local update rule.

The following innovative features have been applied to the basic ACO formulation:

i) pheromone based list-scheduling on a digraph;

ii) trade-off of the α and β parameters (Table 1), which strongly reduces the number of ants of the colony (selected *colony_size* = 8);

iii) computation efficiency induced by the selected low *colony_size* (Table 2), where each ant visits the digraph and produces the minimum computation time, O($m \cdot n$);

iv) freezing condition in the ant path generation, described in section 4.2, which favors the exploration in the initial part of the digraph and then favors the exploitation activity instead of letting it explore constantly. This mechanism represents a compromise between exploring all possible paths (high computation time) and focusing only on the best ones (stagnation in local optima). This mechanism produces better solutions in less computation time;

v) effective mechanism to generate the candidate list, described in section 4.3, based on a parameterization of the non-delay schedules, which improves the algorithm convergence;

vi) use of the stability condition, described in 4.7, which has the expected benefit of finding better solutions by additional iterations, while there are improvements, and consequently it improves the repeatability of the algorithm on different runs. This benefit has been experimentally found. The standard deviation versus the average of the makespan obtained in the 10 runs ranges from 0.29% for flcmax_40_15_10 to 1.73% for flcmax_50_20_7 only. The main drawback of the stability condition is that the processing time increases and can vary significantly. No correlation has been found between processing time and *%gap* from UB or LB, so the better results do not seem a consequence of the higher processing time.

7 Conclusion

We have defined and described how to build a native non-permutation algorithm and shown how the basic ACO algorithm fulfils this requirement, based on a digraph approach.

A novel ACS with the original mechanisms of list scheduling and with the freezing and the stability conditions has also been proposed.

The better performance have been assessed versus the FS benchmark from Demirkol and make the proposed ACO the new state-of-the-art native non-permutation algorithm.

Future research includes designing other metaheuristics to tackle the non-permutation flowshop problem, by taking advantage of the digraph approach discussed in this paper, including hybridized algorithms and other implementations of ACO, like collaborative or hierarchical ACO.

References

Arnaout, J.-P., Musa, R., & Rabadi, G. (2012) A two-stage Ant Colony optimization algorithm to minimize the makespan on unrelated parallel machines—part II: enhancements and experimentations. *Journal of Intelligent Manufacturing*, doi:10.1007/s10845-012-0672-3

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

Bonabeau, E., Dorigo, M., & Theraulaz, G. (2000) Inspiration for optimization from social insect behaviour. *Nature*, 406, 39–42.

Brucker, P., Heitmann, S., & Hurink, J. (2003) Flow-shop problems with intermediate buffers. *OR Spectrum*, 25, 549–574.

Demirkol, E., Mehta, S., & Uzsoy, R. (1998) Benchmarks for shop scheduling problems, *European Journal of Operational Research*, 109, 137–141.

Dorigo, M. & Gambardella, L.M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1, 53–66.

Garey, M.R., Johnson, D.S., & Sethi, R. (1976) The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2), 117–129.

Giffler, D., & Thompson, G.L. (1960) Algorithms for solving production scheduling problems. *Operations Research*, 8, 487–503.

Jain, A.S., & Meeran, S. (2002) A multi-level hybrid framework applied to the general flow-shop scheduling problem. *Computers & Operations Research*, 29, 1873–1901.

Johnson, S. (1954) Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1, 61.

Kumar, R., Tiwari, M.K., & Shankar, R. (2003) Scheduling of flexible manufacturing system: an ant colony optimization approach. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 217, 1443–53.

Liao, C.J., Liao, L.M., & Tseng, C.T. (2006) A performance evaluation of permutation vs. non-permutation schedules in a flowshop. *International Journal of Production Research*, 44 4297–4309.

Lin, S.-W., & Ying, K.-C. (2009) Applying a hybrid simulated annealing and tabu search approach to non-permutation flowshop scheduling problems. *International Journal of Production Research*, 47(5), 1411-1424. doi:10.1080/00207540701484939

Mirabi, M., Fatemi Ghomi, S.M.T., &·Jolai, F. (2011) A two-stage hybrid flowshop scheduling problem in machine breakdown condition. *Journal of Intelligent Manufacturing*, doi:10.1007/s10845-011-0553-1

Nawaz, M., Enscore Jr., E.E., & Ham, I. (1983) A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The International Journal of Management Science*, 11(1), 91–95.

Nowicki, E., & Smutnicki, C. (1996) A fast taboo search algorithm for the job-shop problem. *Management Science*, 42(6), 797–813.

Pinedo, M. (1995) *Scheduling: theory, algorithms and system*. Prentice-Hall, New Jersey.

Potts, C.N., Shmoys, D.B., & Williamson, D.P. (1991) Permutation vs. non-permutation flow shop schedules. *Operations Research Letters*, 10, 281–284.

Pugazhendhi, S., Thiagarajan, S., Rajendran, C., & Anantharaman, N. (2004) Generating non-permutation schedules in flowline-based manufacturing systems with sequence-dependent setup times of jobs: A heuristic approach. *The International Journal of Advanced Manufacturing Technology*, 23, 64–78.

Pugazhendhi, S., Thiagarajan, S., Rajendran, C., & Anantharaman, N. (2002) Performance enhancement by using non-permutation schedules in flowline-based manufacturing systems. *Computers and Industrial Engineering*, 44(1), 133–57.

Rajendran, C., & Ziegler, H. (2004) Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, 155, 426–38.

Rossi, A., & Dini, G. (2007) Flexible job-shop scheduling with routing flexibility and separable setup time using ant colony optimisation method. *Robotics and Computer Integrated Manufacturing*, 23, 503–16.

Rossi, A., Puppato, A., & Lanzetta, M. (2012) Heuristics for Scheduling a Two-stage Hybrid Flow Shop with Parallel Batching Machines: an Application on Hospital Sterilization Plant. *International Journal of Production Research* (accepted for publication).

Sadjadi, S.J., Bouquard, J.L., & Ziaee, M. (2008) An ant colony algorithm for the flowshop scheduling problem. *Journal of Applied Sciences*, 8(21), 3938-44, issn:1812-5654; doi:10.3923/jas.2008.3938.3944

Stuetzle, T. (1998) An ant approach to the flow shop problem. Proceedings of the Sixth European Congress on Intelligent Techniques and Soft Computing (EUFIT'98), Verlag Mainz, Wissenschaftsverlag, Aachen, Germany, 3, 1560–1564.

Stuetzle, T. & Hoos, H.H. (2000) Max–min ant system. *Future Generation Computing Systems*, 16(9), 889–914.

Tandon, M., Cummings, P.T., & LeVan, M.D. (1991) Flowshop sequencing with non-permutation schedules. *Computers and Industrial Engineering*, 15(8), 601–607.

Tavares Neto, R.R. & Godinho Filho, M. (2013) Literature review regarding Ant Colony Optimization applied to scheduling problems: Guidelines for implementation and directions for future research. *Engineering Applications of Artificial Intelligence*, 26, 150–161.

Vijay chakaravarthy, G., Marimuthu, S., & Naveen Sait, A. (2011) Performance evaluation of proposed Differential Evolution and Particle Swarm Optimization algorithms for scheduling m-machine flow shops with lot streaming. *Journal of Intelligent Manufacturing*, doi:10.1007/s10845-011-0552-2

Werner, F., & Winkler, A. (1995) Insertion techniques for the heuristic solution of the job-shop problem. *Discrete Applied Mathematics*, 58(2), 191–211.

Yagmahan, B., & Yenisey, M.M. (2010) A multi-objective ant colony system algorithm for flow shop scheduling problem. *Expert Systems with Applications*, 37, 1361–1368.

Ying, K.-C., & Lin, S.-W. (2007) Multi-heuristic desirability ant colony system heuristic for non-permutation flowshop scheduling problems. *International Journal of Advanced Manufacturing Technology*, 33, 793–802.

Ying, K.-C., Gupta, J. N.D., Lin, S.-W., & Lee, Z.-J. (2010) Permutation and non-permutation schedules for the flowline manufacturing cell with sequence dependent family setups. *International Journal of Production Research*, 48(8), 2169–2184.