Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2013), *27*, 349–357.
© Cambridge University Press 2013 0890-0604/13 $25.00
doi:10.1017/S0890060413000176

Nonpermutation flow line scheduling by ant colony optimization

ANDREA ROSSI AND MICHELE LANZETTA
Department of Civil and Industrial Engineering, University of Pisa, Pisa, Italy

(RECEIVED May 8, 2012; ACCEPTED February 15, 2013)

Abstract

A flow line is a conventional manufacturing system where all jobs must be processed on all machines with the same operation sequence. Line buffers allow nonpermutation flowshop scheduling and job sequences to be changed on different machines. A mixed-integer linear programming model for nonpermutation flowshop scheduling and the buffer requirement along with manufacturing implication is proposed. Ant colony optimization based heuristic is evaluated against Taillard's (1993) well-known flowshop benchmark instances, with 20 to 500 jobs to be processed on 5 to 20 machines (stages). Computation experiments show that the proposed algorithm is incumbent to the state-of-the-art ant colony optimization for flowshop with higher job to machine ratios, using the makespan as the optimization criterion.

Keywords: Ant Colony System; Benchmark Problems; Manufacturing System; Nonpermutation Flowshop; Scheduling

1. INTRODUCTION

A flow line is a conventional manufacturing system where all jobs must be processed on all machines with the same operation sequence (Fig. 1). Jobs are processed only once by each machine, as opposed to reentrant flow lines.

Examples of flow lines include transfer lines, assembly lines, chemical plants, logistics, and many more (Rossi et al., 2012, in press); the problem is scalable 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; time is measured by nondimensional units and can indicate seconds, hours, days, and so on. Flow line is referred to as the physical layout; flowshop is the mathematical model, as defined in the next chapter.

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 *nonpermutation* flowshop (NPFS) scheduling, the most general flowshop case, which is examined here, the order in which all m machines are visited by the n jobs changes, allowing job sequences to be different on subsequent machines.

In a *permutation* flowshop (PFS), the sequence in which jobs visit machines (routing) is the same for all jobs, as for nonpermutation. The sequence of jobs on all machines is the same in PFS; instead, in NPFS the sequence of jobs can be different on subsequent machines.

To allow nonpermutation, buffers between, on board, or shared among machines are necessary. Examples of buffers are shown in the *U*-shaped flow line of Figure 2: input and output buffers are at the two ends and between machines, cells, lines, or plants; they can be shared, in the form of an automatic warehouse or an open space. To allow permutations, jobs travel through buffers between machines. The flow line in the pictorial example itself is made of flow lines: a transfer line and a flexible cell.

The buffer requirement has been formally included in the proposed model. If buffers are not present, either the *blocking* or the *no-wait* condition should be applied to the algorithm to achieve a feasible schedule. In the former case, a job completed on one machine may block that machine until the next downstream machine is free; in the latter case, the next machine must be available before a job leaves the previous one.

As for the problem complexity, there are $(n!)^m$ different schedules for ordering jobs on machines in NPFS; the number of schedules for PFS reduces to $n!$.

In this work, transport and setup times are neglected. This hypothesis often applies when pallet changing systems on machines and fast transport and buffer loading/unloading devices are present. The processing time can be increased by standard transport and/or setup time, if it is relatively small with respect to the processing time; a transport time to and from the warehouse of hours can be considered negligible if the processing

Reprint requests to: Michele Lanzetta, Department of Civil and Industrial Engineering, University of Pisa, Largo Lazzarino, 56122 Pisa, Italy. E-mail: lanzetta@unipi.it

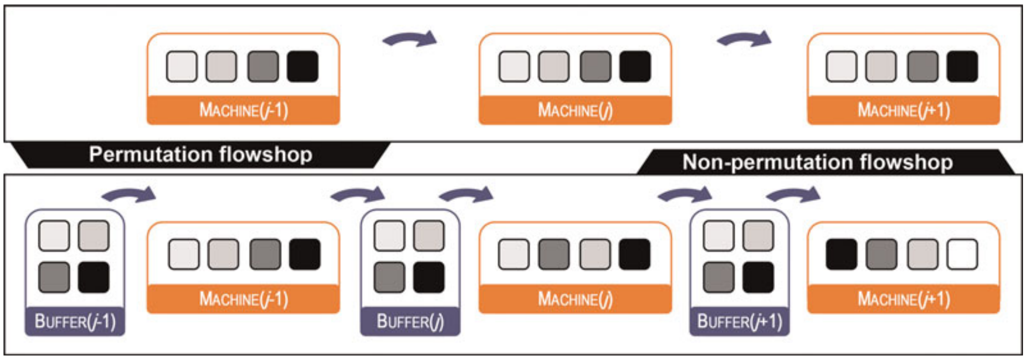

Fig. 1. (Color online) Two flow lines, with and without buffers. Permutation (PFS) and nonpermutation flowshop (NPFS) are compared. In both cases, jobs see machines (routing) in the same sequence (flowshop). In NPFS, buffers allow changes (permutations) of job sequences on subsequent machines.

time is in the order of days, like in the case of welding, heat treatments, painting, and inspection of large and bulky parts.

Operations and transport can be automated, like in computer-integrated manufacturing, or manual. Manual operations can also be represented, using standard times.

Examples of scheduling optimization targets are minimizing total completion time (makespan) or weighted tardiness, balancing mean flow time, and meeting due date. The problem examined here is referred to as $F_m|B_i = +\propto|C_{\max}$ using Graham notation, where F_m stands for flowshop with m machines; $B_i = +\propto$ denotes that buffers with infinite capacity are present, allowing nonpermutation schedules; and $C_{\max}$ denotes the makespan minimization as the optimization criterion. Minimizing the makespan is one of the most common criteria in the literature: lower total completion time is associated with less idle time, higher machine utilization, and efficiency.

Some authors generate random problems or use data taken from realistic cases to test the performance of their proposed algorithms. Demanding benchmark problems allow comparing objectively and quantitatively the performance of different algorithms, also belonging to different classes (e.g., heuristics and metaheuristics). Among the most used flowshop benchmarks is the set by Taillard (1993) considered in this work, which includes small, medium, and large sets, as opposed to Demirkol, whose data set is limited to medium size. Nonpermutation bounds from several authors are available at http://www.mathematik.uni-osnabrueck.de/research/OR/fsbuffer/taillard2.txt (mirrored in http://www.ing.unipi.it/lanzetta/flowshop/taillard2.txt), and they have been included in the current analysis.

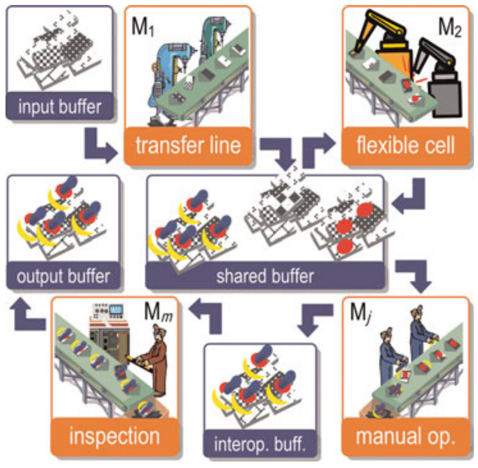

Fig. 2. (Color online) The flow line (clockwise from top left) with m machines (or stages) M (bright red online only) and different examples of buffer configurations (dark blue online only) to allow job sequence permutation between machines.

Biologically inspired general-purpose optimization algorithms are capable of dealing with large job-size problems and with the exponential increase in the solution search space with the number of machines and jobs. Examples of metaheuristics include taboo search, simulated annealing, genetic algorithms (Elbeltagi et al., 2005), and memetic algorithms (Amaya et al., 2012). Despite their successful performance, in the extensive reviews by Ruiz and Maroto (2005) and by Ribas et al. (2010), ant colony or pheromone-based systems are not present. Ant colony systems, a subset class of ant colony optimization (ACO), use artificial or swarm intelligence by exploiting the experience of an ant colony as a model of self-organization in cooperative food retrieval (Wang et al., 2003).

ACO has been selected among metaheuristics because of its ability to build constructively arbitrary permutations of job sequences (NPFS schedules) by two inverse mechanisms: negative and positive pheromone deposition, respectively, through the local update rule and off-line pheromone update rule, detailed in the ACO description. Diversification by the local update rule pushes toward permutated schedules and is the core mechanism to generate natively nonpermutation solutions.

Standard ACO by Bonabeau et al. (1999) and disjunctive graph model inspired by Rossi and Dini (2007) are combined in this paper. It seems that the only ant colony algorithm applied to NPFS scheduling is by Sadjadi et al. (2008), which provides the relative average performance on Taillard's benchmarks. Other NPFS benchmarks from Demirkol have been considered by Ying and Lin (2007) and by the authors (Rossi & Lanzetta, 2013*a*, 2013*b*).

Sadjadi et al. (2008) applied the standard ACO specifications from Bonabeau et al. (1999), except for the diversification mechanism. The other main difference is on the selection of the initial population, which is determined by improving a permutation solution found by heuristics using local search.

Other approaches to the PFS problem tested on benchmarks based on ant colony systems by Rajendran and Ziegler (2004), min–max ant systems by Stuetzle (1998), the state of the art based on tabu search by Brucker et al. (2003), and genetic algorithms by Färber and Coves Moreno (2006) are also compared with the proposed ACO.

2. NPFS PROBLEM

The mixed-integer linear programming model for the NPFS problem is the following:

1. Parameters:

$$p_{ij} = \text{processing time of job } i \text{ on machine } j$$
$$BigM = \text{a sufficiently large positive value}$$

2. Decision variables:

$$Z_{ilj} = 1, \text{ if job } i \text{ is assigned to sequence position } l \text{ on machine } j; \text{ 0 otherwise}$$

3. Dependent variables:

$$S_{lj} = \text{starting time of job in sequence position } l \text{ on machine j}$$

where i and i' are jobs, $i, i' = 1, 2, \ldots, n$; l and l' are the sequence positions, $l, l' = 1, 2, \ldots, n$; j is the machines, $j = 1, 2, \ldots, m$; and the variables n and m are the number of jobs and machines, respectively.

4. Objective function:

$$\text{Min} C_{\max}$$

Subject to the following constraints:

$$\sum_{i=1}^{n} Z_{ilj} = 1 \quad l = 1, \ldots, n \quad j = 1, \ldots, m \tag{1}$$

$$\sum_{l=1}^{n} Z_{ilj} = 1 \quad i = 1, \ldots, n \quad j = 1, \ldots, m \tag{2}$$

$$S_{1j} + \sum_{i=1}^{n} p_{ij} Z_{ilj} \le S_{1(j+1)} \quad j = 1, \ldots, m-1 \tag{3}$$

$$S_{lj} + \sum_{i=1}^{n} p_{ij} \times Z_{ilj} \le S_{l(j+1)} \quad i = 1, \ldots, n-1 \quad j = 1, \ldots, m \tag{4}$$

$$\text{BigM}(2 - Z_{ilj} - Z_{il'(j+1)}) \ge S_{lj} + p_{lj} + S_{l'(j+1)} \quad i, l, l' = 1, \ldots, n \quad j = 1, \ldots, m-1 \tag{5}$$

$$|\langle i, i = 1, \ldots, n, i \neq i': S_{1j} < S_{lj} \times Z_{ilj} \le S_{1j} + p_{i'j} Z_{j'l}, \; l = 2, \ldots, m \rangle| \le n-2 \quad j = 2, \ldots, m \tag{6}$$

Constraint (1) ensures that each job is assigned to exactly one position of the job sequence on every machine. Constraint (2) states that each position of the job sequence processes exactly one job on every machine. Constraint (3) denotes the starting times of the first job on every machine. Constraint (4) insures that the $(l+1)$th job in the sequence of machine j does not start on machine j until the lth job in the sequence of machine j has completed. Constraint (5) insures that the starting time of job I, which is assigned to position l in the sequence on machine $j+1$, is not earlier than its finish on machine j. Constraint (6) ensures that the buffer size is subject to:

Lemma. *The flowshop scheduling with n jobs and m machines is $B_i = +\infty$ if and only if the interoperational buffer size for machine j $(2 \le j \le m)$ is at least $(n-2)$.* ■

The buffer size for machine $j = 1$ and $j = m+1$ is n (i.e., the input and output buffers contain up to n jobs).

Proof: In the worst case, only one blocking with $(n-1)$ jobs waiting occurs. Let j $(2 \le j \le m)$ be the blocked machine. If the last job on machine $(j-1)$ is completed, no blocking occurs because $(n-1)$ jobs have been already processed on machine $(j-1)$. Hence $(n-2)$ jobs wait in the interoperational buffer between machines $(j-1)$ and j. ■

The optimization problem (1)–(6) can also be represented by a disjunctive graph (DG; Fig. 3):

$$\text{DG} = (N, A, E_j, W), \tag{7}$$

where N is the set of operations, plus the dummy start and finishing operations represented by the symbols 0 and *; A is the set of conjunctive arcs (directed arrows) between every pair of operations on a job routing; E_j is the set of disjunctive arcs between pairs of operations at stage j; and W is the set of weights (processing times) on nodes.

3. ACO FOR NPFS

The pheromone trail is the basic mechanism of communication among real ants. It is mimicked by ACO by an iterative method (in epochs) able of finding the shortest path connecting source 0 (nest) and destination * (food) on a weighted graph (Fig. 3), which represents the optimization problem.

The ant runs the nest–food path by a probabilistic selection of nodes according to the following mechanisms: *intensification* to select a node in the vicinity of the current best paths; and *diversification* in order to produce promising alternative paths.

The proposed ACO follows the standard recommendation for applications to scheduling problems, as opposed to the

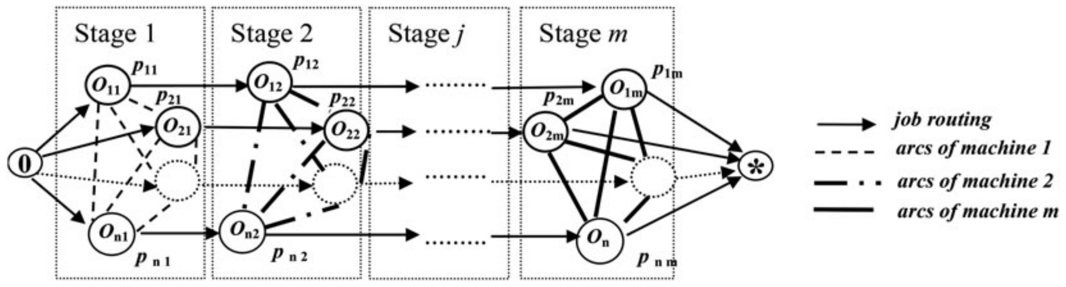

Fig. 3. A disjunctive graph (digraph) for flowshop scheduling, with processing times p_{ij} at nodes O_{ij} for n jobs on m machines.

other implementations available in the literature for the flowshop problem introduced above.

The proposed digraph approach builds *natively* nonpermutation sequences by the *path generation* mechanisms. In this stochastic process, each artificial ant selects probabilistically the next node (move selection) according to the amount of pheromone on the connecting arc (*learned desirability*).

The path associated with each ant starts from 0, follows routing arcs, directs disjunctive arcs, and ends in *. By design, nonpermutation schedules are achieved by directing arcs differently at each stage. Here, $C_{\max}$ is evaluated from W (7). At each epoch, as soon as all the paths of the ants in the colony are generated, the best ant (lowest $C_{\max}$) deposits on its arcs an amount of pheromone proportional to the path length (*pheromone updating*). A pheromone decay routine is also performed to prevent stagnation in local optima solutions (*evaporation* $\rho = 0.12$).

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 toward permutated schedules and is the core mechanism to generate natively nonpermutation solutions.

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 and may take advantage of modern processors.

3.1. Path generation

By the pheromone mechanism, ants may select arbitrary path, consequently the resulting scheduling sequences (ant tours) are different permutations (nonpermutation approach). Random initial solutions are generated and iteratively improved at each epoch by the ant behavior. By this natively constructive approach we are able to assess the net performance of the algorithm.

An ant a to generate an acyclic conjunctive graph with weights on the conjunctive arcs (i.e., feasible schedule S_a), visits every operation on the pheromone-learning model DG (7) one and only one time with a complexity of $O(m \cdot n)$ in order to transform the digraph in a feasible schedule. Path generation is a stochastic process where an ant starts from the dummy 0 and selects the next node from the set of allowed operations. It uses the following *transition probability* rule as a function of both the heuristic function of desirability, η (termed *visibility function*), and the amount of pheromone τ on the edge (O_{ij}, J), with $J \in AL$, of the pheromone trail:

$$z = \begin{cases} \operatorname{argmin}_{o_{ij} \in \mathrm{AL}} \{\tau(o_{i'j}, o_{ij})^{\alpha} \eta(o_{i'j}, o_{ij})^{\beta}\}, & \text{if } q \leq q_0 \\ J, & \text{if } q \leq q_0 \end{cases}. \quad (8)$$

The nonnegative parameters α and β represent the intensity of respectively, the amount of pheromone, and the visibility included in the transition probability function. The nonnegative parameter q_0 is the *cutting exploration*, a mechanism that restricts the selection of the next operation from the candidate list AL. If a random number q is higher than the cutting exploration parameter q_0 ($0 \leq q_0 \leq 1$), the candidate operation is selected by examining the probability of all candidate operations that are as much desirable as higher visibility and pheromone amount are; otherwise, the most desirable operation is selected (i.e., the arc with the highest amount of pheromone and the highest visibility).

The role of cutting exploration is that of explicitly splitting 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. Cutting exploration by tuning parameter q_0 near 1 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 (8). In this paper, a *freezing function* is considered, which is similar to the one proposed by Kumar et al. (2003). This function progressively freezes the system by tuning q_0 from 0 to 1, in order to favor exploration in the initial part of the algorithm and then favor exploitation by means of the following expression:

$$q_0 = \frac{\ln(epoch)}{\ln(n\_epochs)}, \tag{9}$$

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. 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 starting time rule is used, the best one according to Blum and Sampels.

3.2. 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:

$$\tau(O_{i'j}, J) = (1-\rho) \times \tau(O_{i'j}, J) + \rho \times \tau_0. \tag{10}$$

The local update rule is a convex combination of parameters equal to the evaporation coefficient; in this case, the convex combination has points $\tau(O_{ij}, J)$ and τ_0. The amount of pheromone that remains on a selected edge diminishes because it ranges between the previous value $\tau(O_{i'j}, J)$ and the initial value τ_0. As a consequence, 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 pushes the next ants to generate alternative paths.

3.3. Off-line pheromone update rule

This feature arises when a positive amount of pheromone has to be deposited. The ant that detects the best path at each epoch is termed best-epoch ant (S_{be}). In order to direct the exploration of the best nest–food path by the entire colony, an *off-line update rule* of pheromone is performed. At the end of each epoch, the best-epoch ant S_{be} deposits on all paths of the acyclic graph generated a further amount of pheromone, proportional to the following convex combinations of points $\tau(O_{ij}, J)$ and $makespan(S_{\text{be}})^{-1}$. This produces search intensification by other ants of the colony in the vicinity of the best solution:

$$\begin{aligned} \tau'(O_{ij}, J) &= (1-\rho) \times \tau(O_{ij}, J) + \rho \\ &\quad \times makespan(S_{\text{be}})^{-1}, \qquad (O_{ij}, J) \in S_{\text{be}} \\ &= (1-\rho) \times \tau(O_{ij}, J), \qquad \text{otherwise}. \end{aligned} \tag{11}$$

As for the local update rule, the amount of pheromone $\tau'(O_{ij}, J)$ that remains on the selected edge ranges between the previous value, $\tau(O_{ij}, J)$, and a value closer to the optimum: $makespan(S_{\text{be}})^{-1}$. A routine of pheromone decay on pheromone trails is performed on other arcs of the digraph, thus indicating that a path rarely used probably does not lead to optimal solutions.

3.4. Pseudocode

The Ant Colony System for NPFS has been implemented in C++ according to the following high-level description:

```
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

    $W_E(O_{i'j'}, O_{ij}) = (t_0, \tau_0)$ where $\tau_0 = \left[n \times m \times \max_{j=1,\dots,m} \sum_{i=1}^{n} t(O_{ij})\right]^{-1}$

$epoch \leftarrow 1$; $not\_improve \leftarrow 0$;
// Main Loop
while ($not\_improve < stability\_condition$) do
  // Epoch Loop
  for each ant $a$, $a = 1$ to $population\_size$ do
    // Path Generation
    $S_a \leftarrow \emptyset$;
      1. $\boldsymbol{O} \leftarrow \{O_{ij} | i = 1, \dots, n, j = 1, \dots, m\}$;
      2. Initialization of candidate nodes: $AL_w \leftarrow \boldsymbol{O}$;
    for each $w = 1$ to $n \times m$ do
      3. Initialization of feasible moves (i.e., the disjunctive arcs connected to operations of $AL_w$);
      4. Move selection: select a feasible move $(O_{i'j}, O_{ij})$ of $E_A$ where $O_{i'j}$ is the last operation in the queue of machine $m$ ($O_{i'j}$ = dummy 0, if $m = 1$) by means of the transition probability rules (8); directing the related disjunctive arc ($O_{i'j}$ = dummy 0, if $m = 1$);
      5. Arc removal: remove all the remaining disjunctive arcs connected to $O_{i'j}$ (i.e., no other operation can be immediately subsequent to $O_{i'j}$ at stage $j$);
      6. Computing length: move $t(O_{ij}) \in W_N$ from the selected node to the directed one; also, move $t(O_{ij})$ on $(O_{i(j-1)}, O_i) \in A$;
      7. Path length evaluation: the longest path between the one connected to the directed arc and the one connected to the arc of the job routing is placed as a mark of the scheduled operation;
      8. Local updating: apply the local update rule (10) to the arcs $(O_{i'j'}, O_{ij}) \in W_E$;
      9. Update allowed list: remove the scheduled operation from the allowed list, $AL_w \leftarrow AL_w / \{O_{ij}\}$;
    end for
```

```
        10. Directing the remaining disjunctive arcs (i.e., the
            arcs are connected to dummy *).
        11. Local search: Apply local search to $S_a$ with neighbor structure from Nowicki and Smutnicki (1996);
        12. 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 (11);
    Best ant evaluation:
      if ($makespan(S_{be}) < makespan(S^*)$)
      then (($makespan(S^*) \leftarrow makespan(S_{be})$; $S^* \leftarrow S_{be}$
      and $epoch \leftarrow 0$) and $not\_improve \leftarrow 0$;
      else $epoch$ ++ and $not\_improve$ ++;
      end if
end while
Output: $S^*$
```

4. COMPUTATION EXPERIMENTS

Benchmark instances are arrays $b_{n\times m}$. The $n\times m$ operations of each job on all m machines are represented by their processing times ordered by routing. Taillard's (1993) benchmarks include 12 sets of 10 instances for job numbers $i = 20, 50, 100, 200, 500$ and machine numbers $j = 5, 10, 20$. Each benchmark instance k includes a nontrivial lower (LB_{ijk}) and upper bound (UB_{ijk}). The lower (upper) bound is the maximum (minimum) known theoretical minimum (maximum) attainable makespan. The upper bound can be reduced by new, improved solutions. If it coincides with the lower bound, the optimum for benchmark b_{ijk} has been reached.

Metrics for algorithm performance are the individual relative distances from the upper bound of benchmark instances $b_{n\times m}$ or the mean relative error (MRE) in each set (i, j):

$$\mathrm{MRE}_{ij}^{\mathrm{best}} = \frac{1}{10}\sum_{k=1}^{10}\frac{C_{\mathrm{best}_{ijk}} - \mathrm{UB}_{ijk}}{\mathrm{UB}_{ijk}} \tag{12}$$

$$\mathrm{MRE}_{ij}^{\mathrm{avg}} = \frac{1}{10}\sum_{k=1}^{10}\frac{C_{\mathrm{avg}_{ijk}} - \mathrm{UB}_{ijk}}{\mathrm{UB}_{ijk}} \tag{13}$$

where the best and the average makespan solutions for each set (i, j) of 10 benchmark instances $k = 1, \ldots, 10$ are $C_{\mathrm{best}_{ijk}}$ and $C_{\mathrm{avg}_{ijk}}$, respectively.

The proposed NNP-ACO has been run 10 times with the (selected) parameters in Table 1 on 3-GHz 32-bit Intel Pentium IV based PCs with 2 GB RAM.

The main ACO parameters described and summarized in Table 1 have been derived from the job shop application in Rossi and Dini (2007) and have been explored in preliminary tests with the values indicated for *population_size*, α, β, ρ, and η.

As for the population size, fewer ants have been used compared to Rajendran and Ziegler (2004; 40 ants) and compared to Sadjadi et al. (2008; 1000 ants) in order to reduce the processing time. Consequently, the evaporation rate has been reduced compared to Sadjadi et al. ($\rho = 0.9$) to reduce the effect of random search.

The stop criterion from Sadjadi et al. (2008) is a fixed computation time. Instead we use a stability condition, corresponding to 3000 epochs with error reduction of at least one processing time unit.

4.1. Results

The average performance ($\mathrm{MRE}^{\mathrm{avg}}$) of the proposed ACO are compared in Figure 4 within the same class of problems with

Table 1. *Preliminarily tested and selected parameters for the proposed NNP-ACO*

	NNP-ACO	
Parameter	Tested	Selected
population_size	5, 10, 20	5
τ_0	$\left[n \times m \times \max_{j=1,\ldots,m} \sum_{i=1}^{n} t(O_{ij})\right]^{-1}$	$\left[n \times m \times \max_{j=1,\ldots,m} \sum_{i=1}^{n} t(O_{ij})\right]^{-1}$
α	0.1, 0.2, 0.5, 1, 1.5, 2	2
β	$(0.1 \times i)$, $i = 1, \ldots, 8$	0.3
Stop criterion	*not_improve* < *stability_condition*	*not_improve* < *stability_condition*
stability_condition	3000	3000
q_0	$\frac{\ln(not\_improve + 1)}{\ln(stability\_condition)}$	$\frac{\ln(not\_improve + 1)}{\ln(stability\_condition)}$
ρ	$(0.04 \times i)$, $i = 1, \ldots, 9$	0.12
Local search	Steepest descent (NS)	Steepest descent (NS)
η	EST, PAST (RD)	EST

Note: NNP-ACO, native nonpermutation ant colony optimization; NS, Nowicki and Smutnicki (1996); RD, Rossi and Dini (2007); EST, earliest starting time (dynamic visibility); PAST, precedence-ordering average starting time (static visibility).

Sadjadi et al. (2008), Rajendran and Ziegler (2004), and Stuetzle (1998), which do not provide results for data sets of 200 and 500 jobs.

Although results are discrete, a graphical representation with connecting lines has been preferred to show the separation among the performance of different algorithms.

The differences of makespan of the proposed algorithms of the respective authors have been calculated from the upper bound of the PFS benchmark. Because the detailed values are not available, the proposed algorithm has been compared with the (slightly higher) permutation upper bound for performance assessment.

A lower value of MRE^{avg} means a better performance (lower makespan) of the proposed algorithm compared to the state of the art. A negative value represents a new (lower) upper bound.

For comparison within the same class of algorithms (ACO), MRE^{avg} has been conservatively calculated with respect to the original permutation upper bounds from Taillard (1993), because most available results are for PFS, except Sadjadi et al. (2008).

The MRE^{avg} of the proposed ACO ranges between +0.035 and +0.159, while Sadjadi et al. (2008) is between –0.075 and +1.12, Rajendran and Ziegler (2004) is between +0.72 and +1.86, and Stuetzle (1998) is between +0.196 and +2.475 (not shown). This also means that the MRE^{avg} of the proposed ACO is upper limited to 16% as opposed to 112% from Sadjadi et al. The algorithms in Rajendran and Ziegler (2004) and Stuetzle (1998) show the worst performance overall. Out of scale MRE^{avg} values (available on the respective articles) have not been represented to achieve a higher visualization detail on the best results. The nonpermutation algorithm from Sadjadi et al. behaves clearly better than with the permutation (PFS) constraint. The algorithm from Sadjadi et al. has the best performance with 20 jobs or with 5 machines (small problems). Although the performance on large instances (200 and 500 jobs) are not available from these authors, a degradation of performance with benchmark size (job and machine number) is clearly visible on medium instances. This is enhanced by the steeper trend line for the better (nonpermutation) algorithm from Sadjadi et al.

The upper bounds for the makespan of all 12 sets of 10 benchmark instances in NPFS configuration from various authors and methods, averaged, are reported in Table 2. The MRE^{avg} found by the state of the art from Brucker et al. (2003) and Färber and Coves Moreno (2006), based on tabu search and genetic algorithms, respectively, is compared with the proposed ACO. The better results (highlighted) have been obtained for higher machine numbers and job numbers. Results are not available from Brucker et al. and Färber and Coves Moreno for the 40 largest instances, where the proposed ACO becomes the best known solution.

Here the nonpermutation upper bounds have been used for comparison between the proposed NNP-ACO and the state of the art of metaheuristics in general, using Taillard's (1993) benchmarks.

The last four sets of instances are one order of magnitude more time consuming compared to the other instances (200 compared to 10 min) because of their large size. Other methods use a stop criterion based on a fixed number of epochs or computation time. Instead, we use a stability condition (of 3000 epochs with an improvement of at least one processing time unit), which has been reduced by one order of magnitude and still results in a processing time one order of magnitude higher. By the stability condition instead of a stop criterion, convergence is assured regardless of the epoch number.

4.2. Discussion

As shown, the proposed algorithm becomes the state of the art on the benchmarks used, with the ACO approach, particularly on larger instances.

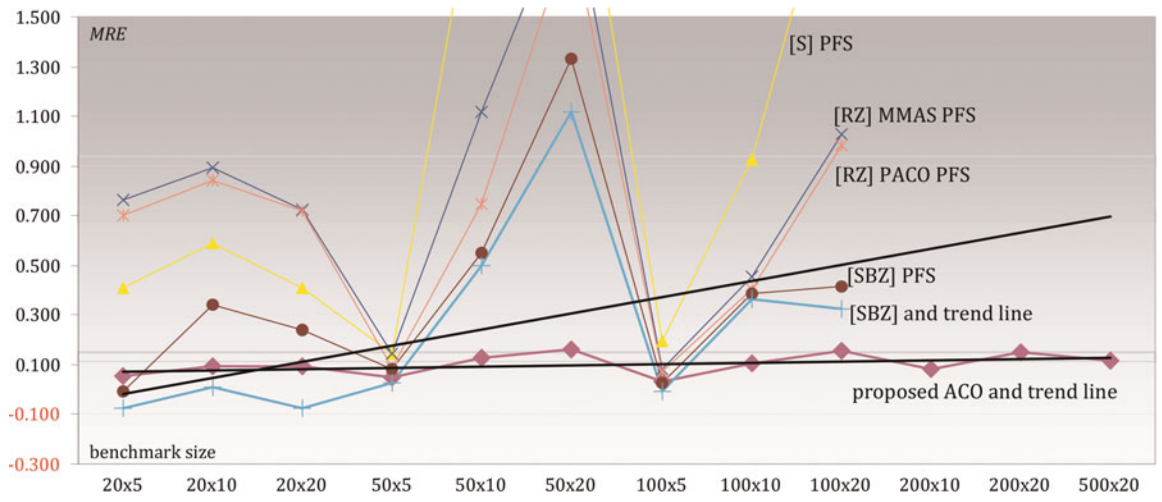

Fig. 4. (Color online) The performance of ant colony optimization (ACO) systems in nonpermutation and in permutation (PFS) configuration on the Taillard's (1993) benchmarks with respect to *permutation* upper bounds from Stuetzle (S; 1998), Rajendran and Ziegler (RZ; 2004), and Sadjadi et al. (SBZ; 2008).

Table 2. *Performance assessment in NPFS configuration*

Instances		Mean NPFS Upper Bounds[a]	State of the Art	Proposed NNP-ACO	
Jobs i	Machines j	Taillard's Benchmarks	MRE^{avg} (12)	MRE^{avg} (12)	MRE^{best} (13)
20	5	1217.1	0.000 (B)	0.057	0.023
20	10	1494.0	0.013 (FCM)	0.107	0.079
20	20	2228.8	0.130 (B)	**0.096**	0.072
50	5	2731.9	0.001 (FCM)	0.048	0.026
50	10	2979.1	0.020 (FCM)	0.136	0.119
50	20	3717.1	0.290 (B)	**0.163**	0.143
100	5	5237.3	0.020 (B)	0.036	0.021
100	10	5618.6	0.130 (B)	**0.107**	0.081
100	20	6312.4	—	**0.165**	0.141
200	10	10663.1	—	**0.084**	0.064
200	20	11272.8	—	**0.160**[b]	0.149[b]
500	20	26362.8	—	**0.120**[b]	0.116[b]

Note: NPFS, nonpermutation flowshop; MRE, mean relative error; $C_{\text{best}_{ijk}}$, the best makespan obtained by the proposed ant colony optimization (ACO) in a single run or otherwise defined by Brucker et al. (B; 2003) and Färber and Coves Moreno (FCM; 2006); NNP, native nonpermutation.
[a]From http://www.mathematik.uni-osnabrueck.de/research/OR/fsbuffer/taillard2.txt
[b]300 epochs.

Possible reasons of the better performance compared to Sadjadi et al. (2008) as a function of the benchmark size are inferred:

1. The ACO implementation by Sadjadi et al. (2008) has a higher colony size and lower epoch number, which do not allow sufficient differentiation despite their higher evaporation, particularly on larger instances.
2. Sadjadi et al. (2008) find an initial permutation schedule (pheromone trails) by the Nawaz et al. (1983) heuristic. Initial good solutions provide good final solution for small-sized benchmarks. For larger benchmarks, the Nawaz et al. heuristic suffers some performance decrease. Consequently, the ACO search can be trapped in local optima.
3. Sadjadi et al. (2008) start from a solution of the permutation problem and find an NPFS solution by a local search, which causes a further performance decrease.

Table 3. *Summary of benefits and drawbacks of the proposed approach*

Benefits	Drawbacks
General purpose optimization algorithm	Parameters need to be selected (and optimized) by preliminary tests.
Constructive solutions from random initialization: net performance can be assessed	Local optima are found (no global optima).
Relative invariance of performance with problem size/complexity	Further research is required to match the performance of other metaheuristic approaches.

The proposed ACO has also been compared with permutation upper bounds from Rajendran and Ziegler (2004) and still provides better performance, despite the higher problem complexity of NPFS ($n!^m$ compared to $n!$).

A regression analysis has been carried out to assess the effect of the machine and job number on the makespan of the best scheduling found by the proposed ACO. A correlation has been found between MRE^{avg} and machine number at constant job number. This is also qualitatively shown by the periodic MRE^{avg} increase in Figure 4. The same trend also shows the relative independence of the algorithm performance on the job number.

A stronger correlation has been found between computation time and both job number and machine number. The computation time with the proposed ACO, which has not been optimized in this work, is one order of magnitude higher than Sadjadi.

Compared to nonpermutation metaheuristics, new upper bounds have been proposed on larger instances and there is still a margin of improvement by parameters optimization. A summary of benefits and drawbacks of the proposed approach is available in Table 3.

5. CONCLUSIONS

A mathematical model of the flow line scheduling problem with buffers has been proposed. The few existing approaches have been compared using well-known benchmarks characterized by a wide size range, available from Taillard (1993).

The NP-hardness has been tackled by metaheuristics, and ACO have been selected. The proposed ACO is natively nonpermutation as opposed to other authors who apply a local search to permutation solutions. Natively means that initial ant paths are selected arbitrarily and the pheromone mechanism stimulates differentiation among permutated schedules (nonpermutation scheduling). The proposed approach shows the best performance in NPFS configuration, particularly on larger instances and is very close to the state-of-the-art metaheuristics.

Based on computation experiments, it can be concluded that such a general-purpose optimization tool has high potential in NPFS and can provide good solutions, regardless of the problem complexity increase in the examined range.

REFERENCES

Amaya, J.E., Cotta, C., & Fernández-Leiva, A.J. (2012). Solving the tool switching problem with memetic algorithms. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 26(2)*, 221–235.

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. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. New York: Oxford University Press.

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

Elbeltagi, E., Hegazy, T., & Grierson, D. (2005). Comparison among five evolutionary-based optimization algorithms. *Advanced Engineering Informatics 19(1)*, 43–53.

Färber, G., & Coves Moreno, A.M. (2006). Benchmark results of a genetic algorithm for non-permutation flowshops using constrained buffers. *10th Int. Research/Expert Conf., Trends in the Development of Machinery and Associated Technology, TMT 2006*, Barcelona-Lloret de Mar, September 11–15.

Kumar, R., Tiwari, M.K., & Shankar, R. (2003). Scheduling of flexible manufacturing system: an ant colony optimization approach. *Journal of Engineering Manufacture 217*, 1443–1453.

Nawaz, M., Enscore, Jr., E.E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA 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.

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*, 426–438.

Ribas, I., Leisten, R., & Framinan, J.M. (2010). Review and classification of hybrid flowshop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research 37*, 1439–1454.

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*, 503–516.

Rossi, A., & Lanzetta, M. (2013*a*). Native metaheuristics for non-permutation flowshop scheduling. *Journal of Intelligent Manufacturing*. Advance online publication. doi:10.1007/s10845-012-0724-8

Rossi, A., & Lanzetta, M. (2013*b*). Scheduling flow lines with buffers by ant colony digraph. *Expert Systems with Applications*. Advance online publication. doi:10.1016/j.eswa.2012.12.041

Rossi, A., Pandolfi, A., & Lanzetta, M. (in press). Dynamic setup rules for hybrid flowshop scheduling with parallel batching machines. *International Journal of Production Research.*

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*. Advance online publication. doi:10.1080/00207543.2012.737942

Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research 165*, 479–494.

Sadjadi, S.J., Bouquard, J.L., & Ziaee, M. (2008). An ant colony algorithm for the flowshop scheduling problem. *Journal of Applied Sciences 8(21)*, 3938–3944.

Stuetzle, T. (1998). An ant approach to the flow shop problem. *Proc. 6th European Congr. Intelligent Techniques and Soft Computing (EUFIT'98)*, pp. 1560–1564, Verlag Mainz, Wissenschaftsverlag, Aachen.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research 64*, 278–285.

Wang, J.F., Liu, J.H., Li, S.Q., & Zhong, Y.F. (2003). Intelligent selective disassembly using the ant colony algorithm. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 17(4)*, 325–333.

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.

Andrea Rossi has been a Research Fellow with the Department of Civil and Industrial Engineering at Pisa University since 2000. He received his PhD in industrial automation and robotics (1998) and his MS in computer science (1993) from Pisa University. He is a referee for international journals and a member of the Italian Association of Mechanical Technology. Dr. Rossi's main interests in manufacturing processes and systems include flexible manufacturing systems production planning and scheduling; inspection planning with coordinate measuring machines; and e-manufacturing and new solution techniques of artificial intelligence, particularly ACO, genetic algorithms, and parallel metaheuristics.

Michele Lanzetta has been an Associate Professor in engineering at Pisa University since 2005. Prior to that, he was an Assistant Professor (1998) at the same institution, where he received his PhD in industrial automation and robotics (1997) and his MEng in aeronautical engineering (1992). He has been visiting scholar at MIT (2011, 2009, 2002, 2000), Stanford University (2007, 2004, 1996), and Tokyo University (2006). Dr. Lanzetta's main interests in manufacturing processes and systems include scheduling, metrology, rapid prototyping, and assembly and visual inspection, particularly in leather and stone. He patented a reflectometer for stone polishing (commercialized) and has published more than 100 papers. He is a Cirp associate (2007–) and Aitem (1995–) member.