Author's accepted manuscript of
Rossi, A.; Lanzetta, M.: Scheduling Flow Lines with Buffers by Ant Colony Digraph, Expert Systems with Applications, vol. 40, n. 9, July 2013, pp. 3328-3340 (13), ISSN: 0957-4174, DOI:10.1016/j.eswa.2012.12.041.
http://www.sciencedirect.com/science/article/pii/S0957417412012821

Scheduling Flow Lines with Buffers by Ant Colony Digraph

Andrea Rossi, Michele Lanzetta

Department of Civil and Industrial Engineering, University of Pisa, Italy

Abstract

This work starts from modeling the scheduling of *n* jobs on *m* machines/stages as flowshop with buffers in manufacturing. A mixed-integer linear programming model is presented, showing that buffers of size *n*-2 allow permuting sequences of jobs between stages. This model is addressed in the literature as non-permutation flowshop scheduling (NPFS) and is described in this article by a disjunctive graph (digraph) with the purpose of designing specialized heuristic and metaheuristics algorithms for the NPFS problem. Ant Colony Optimization (ACO) with the biologically inspired mechanisms of learned desirability and pheromone rule is shown to produce *natively* eligible schedules, as opposed to most metaheuristics approaches, which improve permutation solutions found by other heuristics. The proposed ACO has been critically compared and assessed by computation experiments over existing native approaches. Most makespan upper bounds of the established benchmark problems from Taillard (1993) and Demirkol *et al.* (1998) with up to 500 jobs on 20 machines have been improved by the proposed ACO.

Keywords: Flexible Manufacturing Systems (FMS), Native Non-Permutation Flowshop Scheduling (NPFS), Metaheuristics, Swarm Systems, Ant Colony System (ACS), benchmark problems

1. Introduction

A flow line is a conventional manufacturing system where all jobs must be processed on all machines with the same operation sequence (**Figure 1**). Examples of flow lines include transfer lines, assembly lines, chemical plants, logistics, and many more (Rossi et al., 2010). 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 to optimize one or more objective functions. The job sequence of each machine remains unchanged in a permutation flowshop (PFS) scheduling, while in a non-permutation flowshop (NPFS) the sequence of jobs can be different on subsequent machines.
The impact of buffers, along with machines, products and operations, on reconfiguration and performance evaluation is examined by Colledani and Tolio (2005).

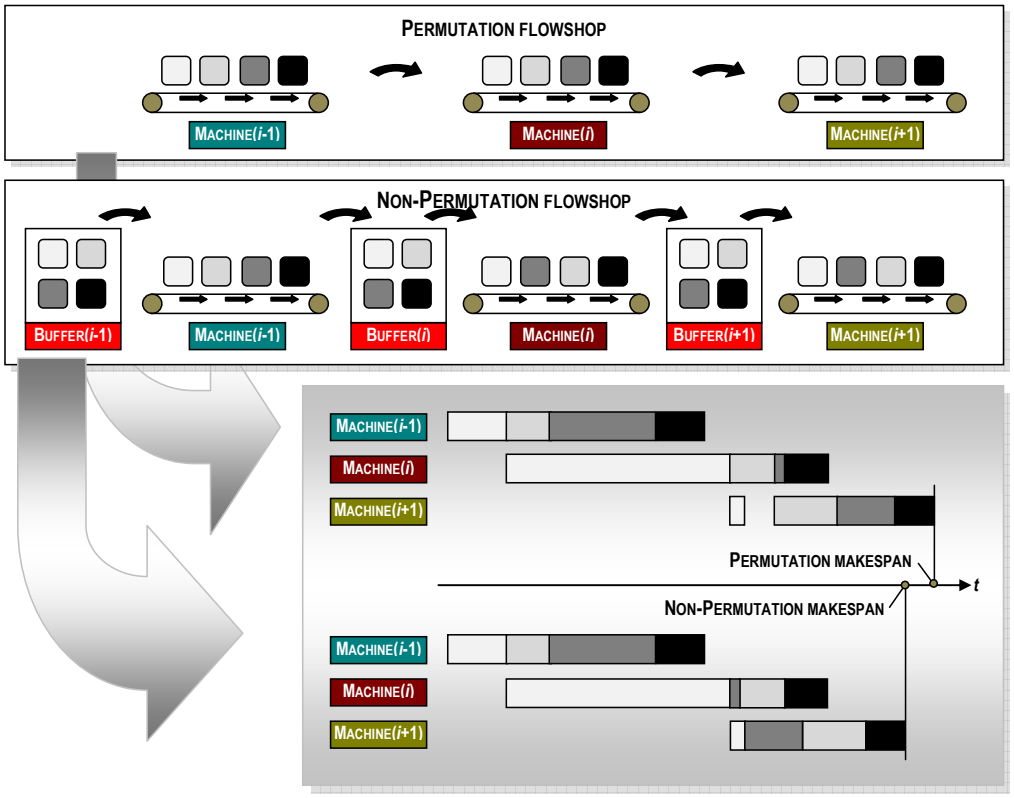

Figure 1. Job Flowing: Permutation Flowshop (PFS) compared to Non-Permutation Flowshop (NPFS) on a physical flow line and on a Gantt diagram.

In permutation flowshop no buffers are present. To achieve a feasible PFS schedule either the *blocking* or the *no-wait* condition should be applied. 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. Examples of flow line, which restrict feasible schedules to PFS schedules are factories with conveyors between machines for material transfer and assembly lines performing the final assembly of bulky products or rigid transfer lines.

In non-permutation flowshop scheduling systems, interoperational or intermediate buffers between two stages are necessary and job passing is allowed (**Figure 1**). Buffers can be shared in the form of an automatic warehouse or an open space. In most manufacturing applications buffers are present, consequently NPFS has a wider interest.

PFS scheduling is an NP-complete combinatorial optimization problem already for $m = 3$ machines (Garey et al. 1976). The computation complexity of permutation approaches is much lower compared with its non-permutation counterpart: there are $n!$ different schedules for ordering jobs on machines in PFS, whilst the number of NPFS schedules increases to $(n!)^m$. To reduce the computation time, permutation flowshop approaches are often preferred by the shop-floor manager, often applying heuristic methods to generate good solutions for practical purposes. PFS scheduler is easier to implement but unfortunately, this simplicity is bought at the price of drastically inferior schedules.

Using the Graham's notation described by a triplet $\alpha|\beta|\gamma$, where field α denotes the system layout and the production flow type, field β indicates the operation characteristics and field γ denotes the adopted performance indices. The problem considered here is $F_m|B_i{=}n\text{-}2|C_{\max}$, where F_m stands for flowshop with m machines, B_i denotes the presence of buffers; a buffer of unlimited capacity ($B_i{=}{+}\infty$) allows permutations, and consequently implies that the NPFS model is applied; a limited capacity $B_i = n\text{-}2$ is sufficient to fulfill the previous requirement, because non-permutation schedules are allowed if the capacity requirement of n-2 is satisfied, as it will be proven in § 3., along with a mixed-integer linear programming model; $C_{\max}$ denotes the makespan minimization as the optimization criterion. Lower makespan implies other benefits, such as lower idle time, higher machine utilization and higher efficiency.

It is well-known that the elimination of buffers has the benefit of reducing the work-in-process (WIP) and improving the line efficiency according to the lean manufacturing philosophy, however as shown in the Gantt diagram in **Figure 1**, changing the sequence of jobs on different machines can have the benefit of shorter schedules. When buffers between machines are present, non-permutation schedules are likely to outperform their permutation counterparts (Tandon et al., 1991). Permutation schedules are dominant up to the three-machine case. In the general case with m (>3) machines, a permutation schedule is not necessarily optimal anymore (Potts et al., 1991). The analysis of theoretical upper bounds of permutation and non-permutation schedules of the well-known 120 instance benchmarks proposed by Taillard (1993), show that system performance are improved by NPFS scheduling (Vaessens, 1996). Simulations by Weng (2000) show that up to 100 jobs and 20 machines, a buffer sizes $B_i \geq 4$ gives a slight gap in

performance with respect to unlimited buffers (B_i=+∝). A possible interpretation is that efficient permutated schedules include permutations of jobs that are at a distance of 4 positions. Permutations of jobs that are very distant in the sequence produce less efficient schedules (e.g. higher makespan). It has also been found that the buffer size should be proportional to the number of jobs. Consequently it can be observed that non-permutation flowshop is more and more beneficial over permutation with higher job number.

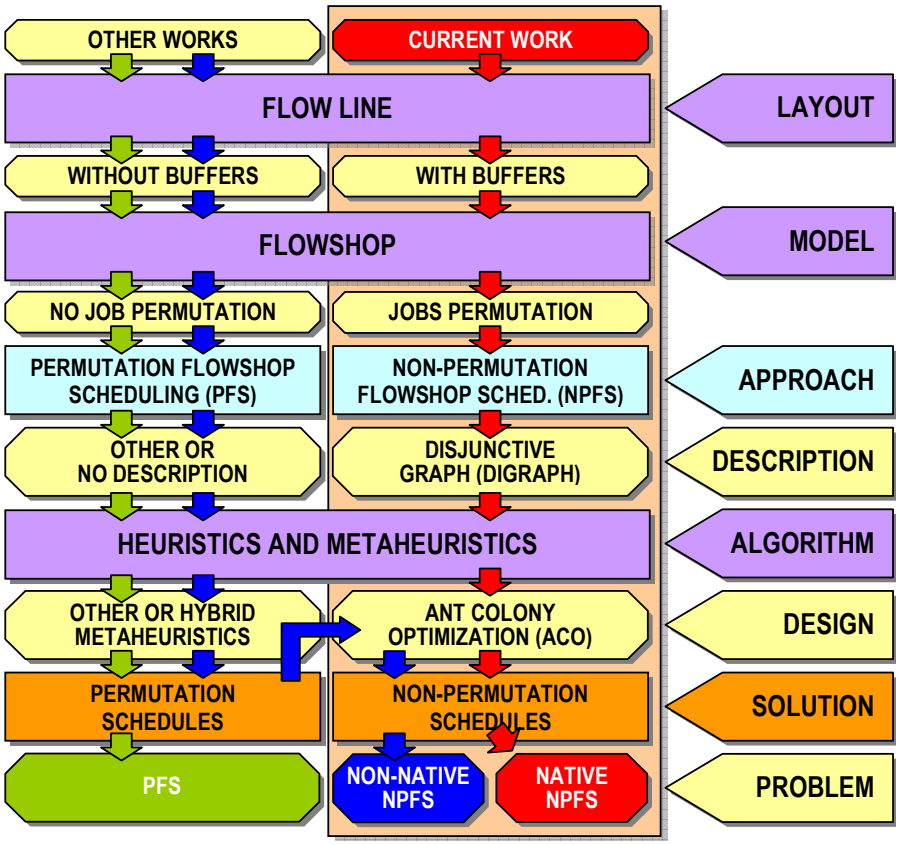

Figure 2. The genesis of native Non-Permutation Flowshop Scheduling (NPFS) and alternatives from the literature to model and approach the flow line problem with and without buffers.

In **Figure 2** the logical flow of current work is summarized with alternatives from the literature, reviewed in the next § 2. It can be noticed that, as anticipated, scheduling on a flowline without buffers can be approached only as permutation flowshop (green path, left); permutation solutions can be used as input for non-permutation algorithms and produce non-permutation schedules (blue path, center). In § 4., the non-permutation approach (formalized in § 3.) is described by a disjunctive graph (digraph) to show how to select or design new heuristics or metaheuristics able to build *natively* non-permutation solutions (red path, right) as opposed to improving permutation solutions found by other heuristics or metaheuristics. The proposed native-NPFS approach (§ 5.) is based on ant colony optimization (ACO) from Bonabeau et al. (2000), as discussed in detail in § 6. Native and non-native approaches are experimentally compared in § 7. considering the benchmarks from Taillard (1990) and Demirkol et al. (1998).

2. Literature

A new class of computation techniques is leading considerable attention: metaheuristic algorithms. Examples of metaheuristics in flowshop scheduling include taboo search (Nowicki and Smutniki, 1996), simulated annealing (Lin and Ying, 2009), immune algorithm (Li et al., 2012), and genetic algorithms (Ruiz et al., 2006). The relevant literature indicates that some of these metaheuristics give considerable results.

Ruiz and Maroto (2005) and Ribas et al. (2010) give a comprehensive review of many heuristic and metaheuristic approaches on permutation flowshop scheduling. A total of 25 algorithms have been coded and tested by Ruiz and Maroto against Taillard benchmarks. As a conclusion, they state that the NEH heuristic from Nawaz et al. (1983) is the best heuristic with Taillard benchmarks. The performance of this heuristic, if implemented as in Taillard is outstanding, with CPU times below 0.5 seconds even for the largest instances.

As opposed to the extensive reviews on PFS, there are few approaches to non-permutation flowshop scheduling (NPFS). Park et al. (1984) adapted a number of heuristics to flowshop scheduling without buffer constrains and concluded that the NEH heuristic is dominant. Afterwards, the metaheuristic performance are improved by Rajendran

and Ziegler (2004) with an ant colony optimization (ACO) approach, Tasgetiren et al. (2007) with a particle swarm optimization method, Zobalas et al. (2009) by a hybrid metaheuristics, which combines the advantages of genetic algorithm with variable neighborhood search, Khalili and Tavakkoli-Moghaddam (2012) by an algorithm risen from the attraction–repulsion mechanism of electromagnetic theories, the multi-objective electromagnetism algorithm, and Ruiz et al. (2006) by a genetic algorithm, which is the current state-of-the-art algorithm tested against Taillard benchmarks.

Among native approaches, a heuristic preference relation, developed as the basis for a heuristic so that only the potential job interchanges are checked for possible improvement with respect to the multicriteria objectives of minimizing makespan and total flowtime, has been applied on Taillard benchmarks by Rajendran (1995). Results have been improved by three native heuristics for the minimization of makespan derived from the minimization of total flowtime based on the sum of idle times on stages by Ravindran et al. (2005), which becomes the better algorithm also for non-native NPFS on Taillard benchmark. Among heuristics, the first native approach based on the shifting bottleneck procedure is from Demirkol et al. (1998), the same authors of benchmark problems.

Koulamas (1998) has reported a new two-phase heuristic where in the first phase, it makes extensive use of Johnson's algorithm. The second phase improves the resulting schedule from the first phase by allowing job passing between machines, i.e. by allowing non-permutation schedules. Leinsten (1990) has proposed heuristics for the flowshop scheduling where passing of jobs between two machines is permitted. The considered system $F_m|B_i \in \mathrm{B}|C_{\max}$ where $\mathrm{B}=\{0,1,..,+\infty\}$ involves a finite buffer capacity, but includes also no buffer capacity and unlimited buffer capacity as a extreme cases.

Tandon et al. (1991) use the enumerative search and a simulated annealing approach to compare PFS and NPFS with the objective of minimizing the makespan. According to their computation experiments, non-permutation schedules usually have global optimum of 2% lower than permutation schedules; benchmarks with random processing times ranging from 1 to 50 (and from 1 to 500) time units were generated. This feature is also treated by Potts et al. (1991), who theoretically proved that there exists a family of instances for which the best permutation flowshop schedule is worse than that of the best non-permutation flowshop schedule by a factor of more than ($m^{0.5}/2$). Pugazhendhi et al. (2003), Liao et al. (2006) and Ying et al. (2010) compare PFS and NPFS with respect to the minimization of a number of flowtime and due-date objectives: total completion time (ΣC_i), total weighted completion time ($\Sigma w_i C_i$), total tardiness (ΣT_i), and total weighted tardiness ($\Sigma w_i T_i$). All the authors use benchmark problems generating random processing time and approach the problem by simulated annealing. In particular, Pugazhendhi et al. (2003) compare PFS and NPFS on $F_m|B_i=+\infty|\Sigma C_i$ and $F_m|B_i=+\infty|\Sigma w_i C_i$ by a heuristic procedure, Liao et al. (2006) evaluate NPFS on $F_m|B_i=+\infty|\Sigma T_i$ by a tabu search and a genetic algorithm, and Ying et al. (2010) consider NPFS by a simulating annealing approach to all the objective functions, $F_m|B_i=+\infty|(C_{\max}, \Sigma C_i, \Sigma w_i C_i, \Sigma T_i, \Sigma w_i T_i)$. Their computation experience shows that all the proposed procedures yield better NPFS schedules than PFS.

Metaheuristic NPFS approaches to minimize flowtime or multicriteria objectives have been proposed by Liao and Huang (2010) on $F_m|B_i=+\infty|\Sigma T_i$ by a tabu search, and by Lin and Ying (2008), who propose three types of metaheuristics, a simulated annealing, a tabu search and a genetic algorithm. They consider sequence-dependent setup times among parts of different batches, $F_m|\mathrm{ST}_{sd,b}, B_i=+\infty|(C_{\max}, \Sigma C_i, \Sigma w_i C_i, \Sigma T_i, \Sigma w_i T_i)$. More recently, Yagmahan and Yenisey (2010) consider a multi-objective criterion $F_m|B_i=+\infty|(w_1 C_{\max} + w_2 \Sigma C_i)$ and propose an ACO. They propose an approach where the initial amount of pheromone on ant trails is a function of the best solution generated by the NEH heuristic (non-native).

Mehravaran and Logendran (2012) consider sequence-dependent setup times and a bicriteria optimization for supply chain customer inventory. Aloulou and Artigues (2010) consider a general flowshop scheduling $F_m|\pi, B_i=+\infty|C_{\max}$ characterized by a partial sequence of jobs on each machine. The purpose of their work is to provide on-line a characterization of possible modifications of a predictive off-line optimal schedule while preserving optimality.

Conversely, few approaches consider the computation experience achieved by benchmarks proposed in literature for the $F_m|B_i=+\infty|C_{\max}$ problem to objectively and quantitatively compare the performance of different algorithms. Ying, Lin and others test various metaheuristic approaches on Demirkol benchmarks: an ant colony system (Ying and Lin, 2007), an iterated greedy heuristic (Ying, 2008) and a hybrid simulated annealing – tabu search method (Lin and Ying, 2009). In their last approach, Lin and Ying use a PFS schedule as the initial solution of their tabu search (non-native).

Brucker et al. (2003) consider $F_m|B_i \in \mathrm{B}|C_{\max}$ where $\mathrm{B}=\{0,1,..,+\infty\}$, a flowshop with buffers, similarly to Leinsten (1990). Computation experiments indicate that CPU times will be reduced and global solutions will be improved by increasing the buffer size. Their tabu search is tested on the Taillard dataset. The Taillard dataset is very effective for $F_m|B_i=0|C_{\max}$ and $F_m|B_i=+\infty|C_{\max}$ because Taillard (1993) and Vaessens (1996) gave a method to evaluate tight upper bounds. Brucker et al. (2003) provide results on NPFS with limited capacity buffer and propose a local search procedure in their tabu search. Their approach is not native: their initial solution is for PFS evaluated by the NEH heuristic; next, their approach becomes non-permutated because the initial job sequences are permutated by local search.

Jain and Meeran (2002) consider a multi-level hybrid approach for the general flowshop scheduling problem $F_m|B_i=+\infty|C_{\max}$. They hybridize a scatter search and a tabu search with the neighborhood structure proposed by Nowicki and Smutnicki (1996) for job-shop scheduling. The initial solution is found by the insertion algorithm proposed by Werner and Winkler (1995).

Tabu search in general requires a good initial solution as input, e.g. produced by outer heuristics, but is unable to generate solutions autonomously: results produced with this technique cannot be used for comparison with native approaches.
ACO has been considered to solve some PFS scheduling systems with the objective to minimize the makespan (Alaykyran et al., 2007; Rajendran and Ziegler, 2004; Ying and Liao, 2004).

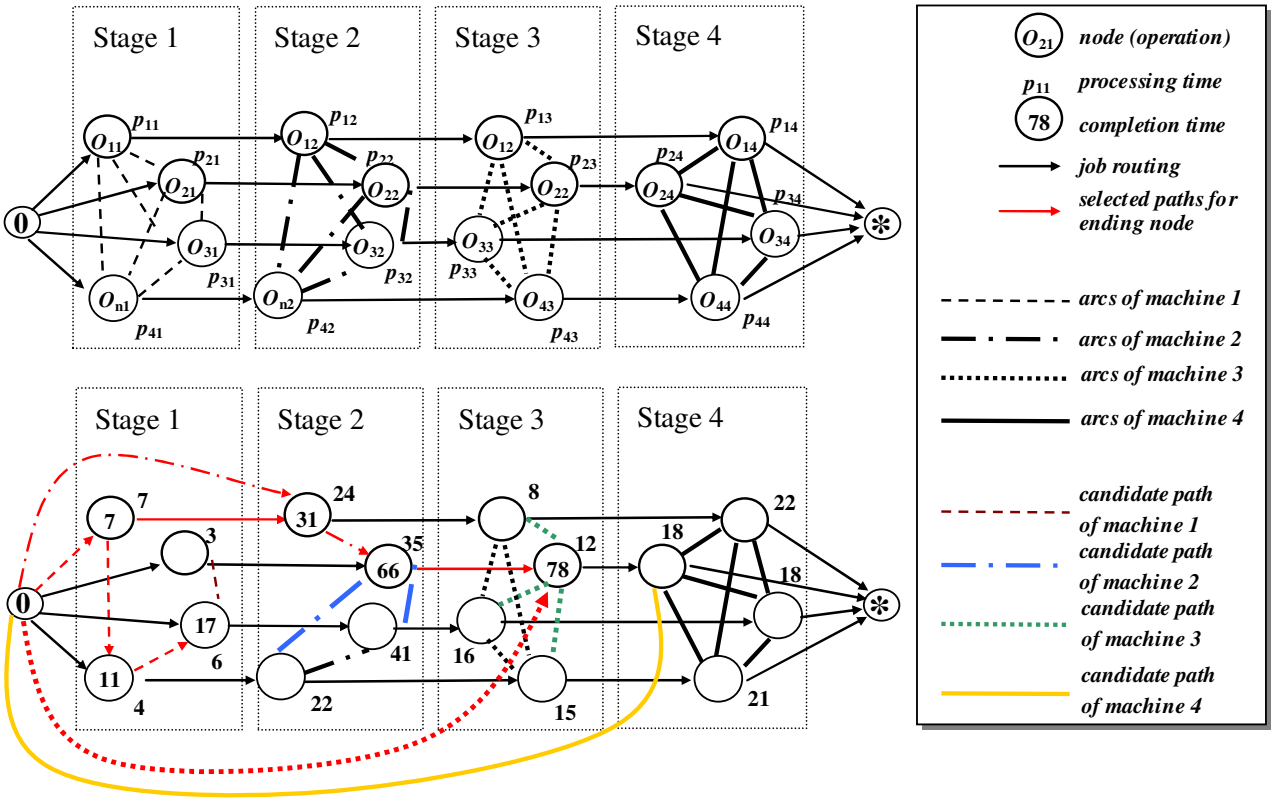

Figure 3. Disjunctive graph (digraph) for flowshop scheduling, with processing times p_{ij} at nodes O_{ij} for n (=4) jobs on m (=4) machines or stages (top). Bottom: an intermediate step of the list scheduling heuristic. Candidate arcs to connect candidate nodes are colored. The completion time of an operation is the longest path that connects dummy 0 and its node.

Figure 3 shows the representation of a feasible schedule by a digraph. In ACO, the digraph is an internal shared memory where, in analogy with nature, artificial ants following trails of pheromone direct disjunctive arcs. Each trail from source and destination nodes represents a possible solution of the problem. Rossi and Dini (2007) describe this mechanism in detail for job-shop scheduling with effective results and integrating a number of features in their digraph: schedules with routing flexibility, sequence-dependent setup and transportation times. Ying and Lin (2007) represent the NPFS problem by a disjunctive graph and use an ant colony system to reach a fully native approach. They adopt a multi-heuristic function of *visibility* based on a set of schedules achieved by dispatching rules. In ACO the visibility of an ant represents the heuristic desirability, that together with the pheromone amount, drives the selection (and the direction) of a disjunctive arc of the digraph to generate a nest-food path. To the best of our knowledge a native-NPFS approach with the objective of minimizing the makespan is applied only by Ying and Lin (2007) on Demirkol benchmark.
The digraph is a general-purpose tool to represent (§ 4.) and design (§ 5.) native NPFS algorithms. The example for an ACO is detailed in § 6.

3. Non-Permutation Flowshop Scheduling (NPFS) problem

This section formalizes the NPFS problem by a mixed-integer linear programming model (MILP), showing its relationship with PFS and the buffer requirement.
In current model, n jobs $i = 1,..,n$ are given, to be processed on m machines $M_1,.., M_m$ and m-1 buffers B_j between machines M_j and M_{j+1} with a capacity of b_j units for $j = 1,.., m - 1$. The buffer size for machine $j = m$ is n (i.e. the output buffer contains up to n jobs). Each job i consists of m operations O_{ij} ($j = 1,.., m$) and operation O_{ij} has to be processed on machine M_j without preemption (job interruption to process higher priority jobs) for $p_{ij} \geq 0$ time units. Between the operations of each job there are precedence relations in the form of a linear routing $O_{i1} \rightarrow O_{i2} \rightarrow .. \rightarrow O_{im}$.

A feasible job schedule is given by assigning start times S_{ij} (and, thus, completion times $C_{ij} = S_{ij} + p_{ij}$) to operations O_{ij} ($i = 1,..,n$; $j = 1,..,m$) such that:

1. the precedence relations between jobs are respected ($C_{ij} \le S_{i+1,j}$),
2. each machine processes only one job at any time ($[S_{ij}, C_{ij}[\cap [S_{kj}, C_{kj}[= 0$ for $j \neq k$),
3. no machine is kept idle when it could start processing some operations (*non-delay* schedule),
4. in each buffer B_i at most n-2 jobs may be stored at the same time, and
5. no blocking conditions are applied.

Blocking occurs when job i completely processed on machine M_j, which could start on M_{j+1} (i.e. $C_{ij} < S_{ij+1}$), finds buffer B_j completely filled by other jobs and machine M_{j+1} is still busy; job i has to wait on machine M_j. While waiting on M_j, job i blocks this machine, so that no other job can be processed on M_j during this time. A job leaving the buffer may be replaced by job i. Since all jobs in buffer B_j have to be processed on machine M_{j+1}, jobs may leave buffer B_j only when machine M_{j+1} becomes available.
The relaxation of the blocking condition at point 5. imposes to job i, which finishes processing on machine M_j, before it can start on M_{j+1} (i.e. $C_{ij} < S_{i,j+1}$), to wait on machine M_j or within buffer B_j during time period $[C_{ij}, S_{i,j+1}]$.

Lemma

The flowshop scheduling with n jobs and m machines is $B_i = +\infty$ if and only if the buffer size for machine j ($1 \le j \le m-1$) is at least (n-2)

■ In the worst case, all the jobs wait for the same machine. Let j ($2 \le j \le m$) be the machine for which the jobs wait for and let i the last job processed on M_{j-1}. Because only non delay schedules are considered (constraint 3.), job i waits on M_j for the machine M_{j+1} which is busy. Hence (n-2) jobs necessarily wait in buffer B_j between machines M_j and M_{j+1} ■

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

l and l' = the sequence positions, l, l' = 1,2,. . .,n

BigM = a sufficiently large positive value

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

Objective function

$$\text{Min } C_{\max} \tag{1}$$

Subject to the following constraints:

$$\sum_{i=1}^{n} Z_{ilj} = 1 \qquad \begin{matrix} l = 1,..,n \\ j = 1,..,m \end{matrix} \tag{2}$$

$$\sum_{l=1}^{n} Z_{ilj} = 1 \qquad \begin{matrix} i = 1,..,n \\ j = 1,..,m \end{matrix} \tag{3}$$

$$\sum_{j=1}^{m} Z_{ilj} \le m \quad \wedge \sum_{j=1}^{m} Z_{i'lj} < m \qquad \begin{matrix} i = 1,..,n,\ i \neq i' \\ l = 1,..,n \end{matrix} \tag{4}$$

$$S_{1j} + \sum_{i=1}^{n} p_{ij} Z_{i1j} \le S_{1(j+1)} \qquad j = 1,..,m-1 \tag{5}$$

$$S_{lj} + \sum_{i=1}^{n} p_{ij} \cdot Z_{ilj} \le S_{l(j+1)} \qquad \begin{matrix} i = 1,..,n-1 \\ j = 1,..,m \end{matrix} \tag{6}$$

$$BigM(2 - Z_{ilj} - Z_{il'(j+1)}) \ge S_{lj} + p_{lj} + S_{l'(j+1)} \qquad \begin{matrix} i,l,l' = 1,..,n \\ j = 1,..,m-1 \end{matrix} \tag{7}$$

$$\left| \left\langle i,\ i = 1,..,n,\ i \neq i':\ S_{1j} < S_{lj} \cdot Z_{ilj} \le S_{1j} + p_{i'j} Z_{i'lj},\ l = 2,...,m \right\rangle \right| \le n-2 \qquad j = 2,..,m \tag{8}$$

Constraint (2) ensures that each job is assigned to exactly one position of the job sequence on every machine. Constraint (3) states that each position of the job sequence processes exactly one job on every machine. Constraint (4) states that, considering all the machine sequences, each job can be at most m times in the same position and at least one of these can be less than m times in the same position (*non-permutation constraint*). Constraint (5) denotes the starting times of the first job on every machine. Constraint (6) ensures that the $(l + 1)^{th}$ job in the sequence of machine j does not start until the l^{th} job in the sequence of machine j has completed. Constraint (7) assures 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 (8) ensures that the buffer size for machine j ($2 \le j \le m$) is at least (n-2).

The proposed formulation limits the difference between NPFS and PFS to a single constraint of the MILP, the non-permutation constraint (4). In the PFS formulation, expression (4) is an equality, because each job needs to stay in the same position for all the machine sequences.

4. Digraph approach for non-permutation flowshop scheduling

The digraph in **Figure 3** used to approach this problem is represented by:

$$DG = (N, A, E_j, W_N) \tag{91}$$

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 between every pair of operations on a job routing, between 0 and every first operation on a routing, and between every last operation on a routing and *; E_j is the set of disjunctive arcs between pairs of operations, O_{*j}, that have to be processed on the machine at stage j (j=1,..,m); it also includes disjunctive arcs between 0 and O_{*j}, between O_{*j} and * and between 0 and * for all E_j; W_N is the weight on nodes, represented by the processing time of related operations O_{ij}, $W_N(O_{ij})=t_{ij}$.

Figure 3 (top) shows an example of digraph for flowshop scheduling. Each operation O_{*j} is connected with disjunctive arcs of E_j only, because the related operation can be processed by a machine at stage j. In general, a non-permutation flowshop problem is represented by $n \times m$ operations and an operation is connected with (n+1) disjunctive arcs.

If no cycle is present in a conjunctive graph achieved by directing some disjunctive arcs to include all the operations, the conjunctive graph becomes *acyclic* and related sequences on machines are *feasible schedules*. In an acyclic conjunctive graph, the makespan of the feasible schedule is the length $L(C)$ of the *critical path*, the longest path between the dummy start and finishing operations.

Figure 3 (bottom) shows an example of acyclic conjunctive graph at an intermediate step, achieved by directing disjunctive arcs. Empty nodes are to be selected. The weighted path (full node) is obtained by adding processing times along the selected path and selecting the highest completion time of all arcs that end to it. The completion time of an operation is the longest path that connects dummy 0 and its node.

5. Native Non-Permutation Flowshop Scheduling heuristic

The native non-permutation flowshop algorithm generates an acyclic digraph by visiting every operation one and only one time. Disjunctive arcs connecting any visited node are immediately directed towards the node. A fast $\mathbf{O}(n \times m)$ approach to generate an acyclic digraph is proposed by Rossi and Dini (2007) considering the list scheduler heuristic for the flexible job-shop scheduling. The list scheduler heuristic is originally proposed by Giffler and Thompson (1960). The behavior of the list scheduling heuristic is to connect exactly one ending and one starting arc for every operation by directing ($n \times m$) of the ($n^2 \times m$) initial disjunctive arcs.

The following heuristic designed for the native non-permutation flowshop scheduling modifies the list scheduling heuristic in order to generate a feasible schedule with completion times of all operations assigned to nodes.

A Native Non-Permutation Flowshop Scheduling (native-NPFS) heuristic

Input: a digraph $DG = (N, A, E_M, W_N)$

$O \leftarrow \{O_{ij} \mid i=1,..,n; j=1,..,m\}$

for each $w = 1$ **to** $|O|$ **do**

1. *Initialization of Candidate Nodes*: build the *allowed list* AL_w for current step w: $AL_w \leftarrow \{O_{ij} \in O \mid O_{ij-1} \cap O = \varnothing\}$
2. *Restriction*: restriction of the allowed list by means of optimality criteria (i.e. active or non-delay schedule); let the *candidate list* CL_w be the restricted allowed list
3. *Initialization of Feasible Moves*: mark as a *feasible move* each disjunctive arc ($O_{i'j}$, O_{ij}) of E_M where $O_{ij} \in CL_w$ and $O_{i'j}$ is the last operation of the sequence of resource j

4. *Move Selection*: select a feasible move ($O_{i'j}$, O_{ij}) of E_M by directing the related disjunctive arc ($O_{i'j}$ =dummy 0, if $j = 1$)
5. *Arcs Removal*: remove all the remaining disjunctive arcs connected to $O_{i'j}$, i.e. no other operation outside of O_{ij} can be immediately subsequent to $O_{i'j}$ at stage j
6. *Transferring weight*: the weight t_{ij} on the node is moved onto the related arc and onto the arc of the job routing (arc of A) that ends at O_{ij}; also, the completion time $t(O_{ij}) = S_{ij} + t_{ij}$ is assigned as a mark inside the node O_{ij}, where the starting time of the operation is the maximum between the completion time of the previous operation in the routing and the previous operation in the sequence at stage j: $S_{ij} = \max\{t(O_{i(j-1)}), t(O_{i'j})\}$
7. *Updating Structures*: update O by removing operation O_{ij}

end for

9. *Directing the remaining disjunctive arcs*: i.e. arcs connected to the dummy operation *

Output: the non-permutation schedule C

The native-NPFS heuristics performs as many steps as the number of nodes (operations). The dummy operation 0 is the start of all machine sequences. For each node, a set of *feasible moves* is initialized. A feasible move is a disjunctive arc that connects the current node, already inserted in the machine sequence, to a next node that represents an operation not already scheduled. A feasible move to connect the related operations is selected by means of the heuristic desirability. This desirability is termed *visibility* (e.g. shortest processing or setup time, nearest due date etc.). Machine sequences are non-permutated because the native-NPFS heuristic allows a random selection of the feasible move and because the selected disjunctive arc can be directed in either the directions. After a feasible move is selected, all disjunctive arcs connected with the starting node are removed to prevent cycles. The weight of the ending node is updated with the maximum distance from the source, evaluated by the following two alternatively paths:

i.) the path that crosses the previous node on the routing;

ii.) the path that crosses the previous operation in the machine sequence.

This guarantees that the weight on the dummy finishing operation is the longest path that starts from the dummy start operation, i.e. the $L(C)$.

6. Proposed Ant Colony Optimization (ACO) algorithm

Ant colony systems, a subset class of ACO, are an emerging class of biologically inspired research dealing with artificial or swarm intelligence: it exploits the experience of an ant colony as a model of self-organization in co-operative food retrieval (Dorigo and Gambardella, 1997).

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.

This section describes an ant colony system (ACS), which makes use of the heuristic designed above by the digraph method and which fulfills all the problem constraints (2) to (8) to generate native non-permutation flowshop schedules.

The proposed native-NPFS ACO based on the list scheduling heuristics described guides ants to select the most desirable move into nest-food path by using the function of visibility (as for native-NPFS heuristics). Pheromone amount laid on the disjunctive arcs mitigate the impact of heuristic desirability because it drives ants to select the optimal move within the set of feasible moves (step 4. of the native-NPFS heuristic). 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 ant runs 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 path C^*.

As soon as all the paths of the ants in the colony are generated, the best epoch ant deposits on the arcs of it path C_{be} a further amount of pheromone proportional to the path length $L(C_{be})$ 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 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 search, to better improve the best found solution.

Local and off-line pheromone update rules are mechanisms of, respectively, ant paths diversification and ant paths intensification. They are implemented by the standard procedures described in Dorigo and Gambardella (1997).

6.1. Digraph model

The disjunctive graph is able to implement pheromone trails. The

$$WDG = (N, A, E_j, W_N, W_E) \tag{10}$$

has the new component W_E, which represents the weight on disjunctive arcs (pheromone amount). The weight on 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 $\tau[O_{i'j}, O_{ij}]$ provides an index of desirability in order to add the feasible move $[O_{i'j}, O_{ij}]$ to the list of jobs to be processed by machine j. The pheromone trail W_E is a two-dimensional array of $[m \times n \times (n+1)]$ elements (considering that the number of disjunctive arcs among all the n operations to be processed on machine j that can be replicated is $[n \times (n-1)/2]$). Hence, the internal shared memory of the proposed ant colony system is $\mathbf{O}(n^2 \times m)$.

6.2. Path generation

To generate a feasible schedule C_a, each ant a visits every operation on *WDG* 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 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 $J \in CL$ and such that

$$Z = \begin{cases} \arg\min_{O_{ij} \in CL} \left\{ \tau(o_{ij}, J)^{\alpha} \cdot \eta(o_{ij}, J)^{\beta} \right\}, & \text{if } q \le q_0 \\ J, & \text{if } q > q_0 \end{cases} \tag{11}$$

The non-negative parameters α and β represent the intensity of respectively, 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 all the candidate operations in probability, proportionally to pheromone amount $\tau(o_{ij}, J)^{\alpha}$ and visibility $\eta(o_{ij}, J)^{\beta}$; 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 explicitly splitting the search space in order to achieve a compromise between the probabilistic mechanism adopted for $q \le q_0$ 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 the second expression in (11). Scheduling problems have many local minima so higher randomization is used when the algorithm starts.
A *freezing function* similar to the 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 according to the expression

$$q_0 = \frac{\ln(epoch)}{\ln(n\_epochs)} \tag{12}$$

where *epoch* is the current iteration and *n_epochs* is the total number of iterations of the ant colony system from the stability condition defined below.
The heuristic function of desirability η is a 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 the proposed ACO the earliest starting time (EST) heuristic, i.e. the best heuristic tested by Blum and Sampels (2004), is used.

6.3. 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: these are schedules in which 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

$$\min_{O_{ij} \in AL} S_{ij} \tag{13}$$

All the operations O^* that can start if no machine is kept idle for more than a predefined value δ verify the condition

$$S_{O^*} \leq \min_{O_{ij} \in AL} S_{ij} + \delta, \qquad O^* \in AL \tag{14}$$

A strategy that relaxes the expression (14) is considered by using the following parametric value of δ

$$\delta(rf) = \frac{\max_{O_{ij} \in AL} S_{ij} - \min_{O_{ij} \in AL} S_{ij}}{rf} \tag{15}$$

where *rf* is the *restricted factor*. If the restriction is maximum, i.e. $rf \to +\infty$, the predefined value $\delta(rf)$ 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

$$CL = \left\{ O^* \in AL \mid S_{O^*} \leq \min_{O_{ij} \in AL} S_{ij} + \frac{\max_{O_{ij} \in AL} S_{ij} - \min_{O_{ij} \in AL} S_{ij}}{rf} \right\} \tag{16}$$

where a restricted factor $rf = 3$ from the cited literature is selected.

6.4. Local search

When a path is generated, the solution is taken to its local optimum by a *local search* routine. The mechanism considered is a steepest descent algorithm where the current best solution is replaced with an improved solution included in the neighbor of current best. The performance of the local search depends on the neighborhood structure, and has been derived from the state-of-the-art local search for job-shop scheduling: the neighborhood structure proposed by Nowicki and Smutnicki (1996) for their fast tabu search algorithm.

6.5. Stability condition

In iterative algorithms, the stop criterion is sometimes a fixed epoch number or computation time. Instead, to stop the algorithm we use a stability condition, corresponding to a fixed number of epochs (3000), with error reduction of at least one makespan time unit per epoch.

6.6. The native approach

A metaheuristic algorithm is considered as a *native approach* if the initial solution is a feasible solution achieved by dispatching rules. Generally, a dispatching rule is selected within a complete set of known rules. Recently, the behavior of an algorithm to be native is considered in computation science to demonstrate the superiority of the single component algorithm of hybrid metaheuristic under study. A native-NPFS approach builds autonomously an initial feasible schedule, which includes different permutations of jobs between machines. Starting with a native (inner) initial sequence, the unbiased performance of the system can be evaluated, because they are not affected by the performance of (outer) heuristics.
According to this definition of native, hybrid approaches cannot be considered native.
A number of metaheuristics approaches are tailored on the digraph. A feasible native schedule can be built by a disjunctive graph approach. Biologically inspired general-purpose optimization algorithms are capable to deal with large job-size problems and with the exponential increase in search space with the number of machines and jobs. In this paper ant colony optimization is considered, because ACO uses a basic mechanism (*diversification*) to generate non-permutated sequences of jobs. The proposed digraph approach builds *natively* non-permutation 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*). By design, non-permutation schedules are achieved by directing arcs differently at each stage. C_{max} is evaluated from the weights on the critical path. 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 towards permutated schedules and is the core mechanism to generate natively non-permutation solutions.

6.7. 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** *ps* **do**

// Path Generation

$C_a \leftarrow \varnothing$;

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

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

3. *Restriction: restrict* AL_w to the candidate list CL_w by means of expression (16);
4. *Initialization 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_{ij}) of E_j, where $O_{ij} \in CL$, by means of the transition probability rules (11); 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 $L(O_{ij})$ of the longest path that connects O_{ij} to the dummy 0 is evaluated by $L(O_{ij}) = t(O_{ij}) + \max\{L(O_{i'j}), L(O_{(i-1)j})\}$ and is placed as a mark near the scheduled operation;
8. *Local Updating*: apply the local update rule to the arcs ($O_{i'j}$, O_{ij}) $\in W_E$ as in Dorigo and Gambardella (1997);
9. *Update Allowed List*: remove the scheduled operation from the allowed list

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

   $\leftarrow AL_w / \{O_{ij}\}$, otherwise;

end for

10. Directing the remaining disjunctive arcs (these arcs are connected to dummy *)
11. *Local Search*: Apply the local search with the neighbor structure from Nowicki and Smutnicki (1996) to C_a;
12. *Best Evaluation*: **if** [$L(C_a) < L(C_{be}$]

then [$L(C_{be}) \leftarrow L(C_{be})$ **and** $C_{be} \leftarrow C_a$]

end if

end for

Global Updating: Apply the global update rule as in Dorigo and Gambardella (1997);

Best Ant Evaluation: **if** [$L(C_{be}) < L(C^*)$]

then [$L(C^*) \leftarrow L(C_{be})$; $C^* \leftarrow C_{be}$ **and** $epoch \leftarrow 0$] **and**

$not\_improve \leftarrow 0$;

else $epoch$ **++ and** $not\_improve$ ++;

end if

end while

Output: C^*

The main parameters of the two ACO approaches are listed in Table 1. The parameters considered in other ant colony systems for NPFS are also listed.

Table 1. Parameters from different ACO approaches, including the proposed native Non-Permutation Flowshop Scheduling ACO.

Parameter	Yagmahan and Yenisey (2010)	Ying and Lin (2007)	native-NPFS ACO
problem	non-native	native	native
population size	20	5	8
candidate list	15	-	self complied (*parameterized non-delay*)
α	2	0.1	2
β	0.5	2.0	0.3
local search	Adjacent Pair-wise Interchange	-	Borderline Critical Path Blocks Interchange
q_0	0.9	0.1	self complied (*freezing function*)
ρ	0.2	0.9	0.12
τ_0	$[n \times C_{NEH}]^{-1}$	$[n \times m \times UB]^{-1}$	$\left[n \times m \times \max_{j=1,..,m} \sum_{i=1}^{n} t(O_{ij}) \right]^{-1}$
η	SPIRIT rule	Multi-Heuristic Desirability	Earliest Finishing Time
stop condition	1000 epochs	5000 epochs	*stability condition*
stability condition (*n_epochs*)	-	-	3000 epochs
firmware platform	2 GHz Intel® Pentium® M760, RAM 1024 MB DDR-2	1.5 GHz Intel® Pentium® 4 RAM N.A.	3 GHz 32 bit Intel® Pentium® 4 RAM 2 GB

7. Computation experiments

7.1. Benchmarks

Benchmark problem sets allows researchers to compare their proposed algorithms with those of other researchers using an identical test problem set. Benchmark instances are arrays $b_{n \times m}$ of processing times of each job on all *m* machines ordered by routing originally generated random. Each benchmark can be synthesized by its makespan, i.e. the processing time obtained as the addition of the $n \times m$ operations of each job on all machines ordered by routing (plus idle times). Two reference makespan values are considered by researchers: non trivial lower (LB) and upper bound (UB). The lower (upper) bound is the maximum (minimum) known theoretical minimum (maximum) attainable makespan. A lower bound (LB) can be obtained for each instance by relaxing the capacity constraints on all but one machine, and solving for optimality the resulting single machine problem of minimizing makespan with release and delivery times (Pinedo, 1995). The upper bound can be reduced by improved solutions. If it coincides with the lower bound, the optimum has been reached.

Table 2. Benchmark sets considered.

	Taillard (1993)	**Demirkol et al. (1998)**	**parameter (index)**
job number	20, 50, 100, 200, 500	20, 30, 40, 50	n (i)
machine number	5, 10, 20	15, 20	m (r)
processing times	random[1,99]	random[1,200]	
n to m ratio	1 – 100	1 – 3.3	
# operations ($n \times m$)	100 – 10,000	300 – 1,000	N
# sets ($n \times m$ combinations)	12	8	
# instances per set	10	10	
# instances considered per set	10	5	
# instances total	120	40	
# instances (individual data) considered by other authors for comparison	28, **Table 3**	40, **Table 4**	
# sets (aggregated data) considered by other authors for comparison	3, **Table 5**	8, **Table 5**	
competitors approaches	Rajendran (1995), Ravindran et al. (2005), Yagmahan and Yenisey (2010)	Demirkol et al. (1998), Ying and Lin (2007), Lin and Ying (2009)	
most recent LBs and UBs	Vaessens (1996), mirrored in Lanzetta (2012) + competitors above	see competitors above	

The proposed native heuristic has been tested by computation experiments on well-known benchmark problem sets established by Taillard (1993) and by Demirkol et al. (1998) characterized in **Table 2**. These combinations yield a problem set that is not based on a specific application.

All jobs are available at time zero, and generated by a discrete uniform distribution. The most used flowshop benchmark set is by Taillard and is listed in **Table 3**. All Taillard instances in each set are considered by authors, instead Demirkol et al. ranked the instances in decreasing order of percentage gap, defined in (17), between the upper and lower bounds for each combination of n and m to obtain a more compact and challenging set of test problems. Only the first five instances for each combination were finally presented. Thus, the test bed comprised a total of 40 test instances, which are listed in **Table 4**.

Vaessens (1996) determined the lower bounds for non-permutation for Taillard benchmarks with a method similar to the mentioned Pinedo (1995). As the size and complexity of the instances make exact solutions impractical, Demirkol et al. (1998) solved each instance by five different constructive heuristics and three versions of the shifting bottleneck procedure and reported the highest makespan value obtained as a lower bound in each instance. All algorithms were run on Unix with 50 MHz SUN SPARC server 1000 Model 1104. The upper bound (UB) for each instance was the best solution found by any of the algorithms. The computation time was recorded by the method that provided the best solution. If more than one algorithm found the best solution, then the algorithm with the shortest computation time was selected.

Before this work, it seems that no author has compared the performance of their proposed approach on both sets, so the results of current approach are presented for each set separately.

7.2. Performance measures

The proposed native-NPFS ACO has been run 10 times with the parameters in **Table 1**. The best and average solutions have been reported in **Table 3** and in **Table 4**.

Metrics for algorithm performance are: 1) the number of benchmark instances where the minimum makespan C_{best}^{ta0xy} of the Taillard benchmark instance ta_{0xy}, y=1,..,9; x=0,1,2 is found with respect to the other native approaches; 2) the $n \times m$ aggregate relative distances from the upper bound or the best known solution evaluated by the relative distance between UB and $\sum_{y=1}^{9} C_{best}^{ta0xy}$, $x = 0,1,2$ for each array $n \times m$ of the 5 Taillard benchmarks:

$$\%gap_{best} = 100 \cdot \frac{\sum_{y=1}^{9} C_{best}^{ta0xy} - \sum_{y=1}^{9} LB^{ta0xy}}{\sum_{y=1}^{9} LB^{ta0xy}}, \quad x = 0,1,2 \tag{17}$$

$$\%gap_{average} = 100 \cdot \frac{\sum_{y=1}^{9} C_{average}^{ta0xy} - \sum_{y=1}^{9} LB^{ta0xy}}{\sum_{y=1}^{9} LB^{ta0xy}}, \quad x = 0,1,2 \tag{18}$$

Analogous measure is evaluated for each array $n \times m$ of the 10 Demirkol benchmarks.

Table 3. Benchmark problems by Taillard (1993), state-of-the-art solutions and results of computation experiments for one non-native and three native NPFS approaches. Yagmahan and Yenisey (2010) is an ant colony system, Rajendran (1995) and Ravindran et al. (2005) are two native heuristics and native-NPFS ACO is the ant colony system proposed in this paper. In bold the best performing native algorithm.

Approach		Benchmark	Non-native Approach	Native Approach			
Algorithm →			Yagmahan and Yenisey (2010)	Rajendran (1995)	Ravindran et al. (2005)	native-NPFS ACO	
Instance ↓		UB/Best Known	C_{best} (10 runs)	C_{best}	C_{best}	C_{best} (10 runs)	$C_{average}$
20x5	ta001	1278	1297	1359	1297	**1290**	1293.1
	ta002	1358	1383	1378	**1373**	1389	1389.0
	ta003	1073	1203	1230	1206	**1100**	1112.5
	ta004	1292	1377	1393	1402	**1344**	1352.2
	ta005	1231	1311	1307	1334	**1250**	1258.7
	ta006	1193	1245	1282	1238	**1217**	1224.3
	ta007	1234	1303	1387	1322	**1258**	1259.6
	ta008	1199	1265	1344	1287	**1235**	1242.5
	ta009	1210	1303	1335	1307	**1258**	1275.3
	ta010	1103	1179	1191	1195	**1127**	1145.5
20x10	ta011	1560	1681	1711	1774	**1693**	1729.6
	ta012	1644	1749	1916	1791	**1785**	1799.5
	ta013	1486	1554	1617	1643	**1583**	1596.5
	ta014	1368	1490	1533	1531	**1452**	1478.3
	ta015	1413	1455	1588	1557	**1516**	1526.7
	ta016	1369	1564	1565	1612	**1445**	1468.3
	ta017	1428	1590	1622	1594	**1524**	1544.5
	ta018	1527	1595	1800	**1631**	1650	1663.8
	ta019	1586	1689	1717	1769	**1659**	1681.1
	ta020	1559	1719	1831	1744	**1670**	1677.7
20x20	ta021	2293	2428	2610	2491	**2396**	2414.9
	ta022	2092	2281	2301	2491	**2225**	2239.5
	ta023	2313	2515	2411	**2422**	2446	2464.5
	ta024	2223	2299	2471	2567	**2346**	2360.8
	ta025	2291	2473	2427	2420	2439	2460.5
	ta026	2221	2339	2466	2557	**2331**	2346.5
	ta027	2267	2378	**2174**	2448	2428	2454.0
	ta028	2183	2418	2418	2464	**2321**	2345.6

Table 4. Benchmark problems by Demirkol et al. (1998), state-of-the-art solutions and results of computation experiments for one non-native and three native NPFS approaches. Lin and Ying (2009) is a hybrid simulated annealing – tabu search, Demirkol et al. (1998) is a shifting bottleneck heuristic, Ying and Lin (2007) is an ant colony system and native-NPFS ACO is the ant colony system proposed in this paper. In bold the best performing native algorithm.

Approach	Benchmark	Non-native Approach	Native Approach			
Algorithm →		Lin and Ying (2009)	Demirkol et al. (1998)	Ying and Lin (2007)	native-NPFS ACO	
Instance ↓	LB	C_{best} (5 runs)	C_{best}	C_{best} (5 runs)	C_{best} (10 runs)	$C_{average}$
	flcmax_20_15_3	3354	3873	4437	4420	**4047**	4113.4
	flcmax_20_15_6	3168	3761	4144	4044	**3950**	3977.5
20x15	flcmax_20_15_4	2997	3518	3779	3786	**3692**	3730.6
	flcmax_20_15_10	3420	4051	4302	4265	**4176**	4221.7
	flcmax_20_15_5	3494	3913	4373	4310	**4097**	4124.9
	flcmax_20_20_1	3776	4525	4821	4819	**4790**	4826.9
	flcmax_20_20_3	3758	4435	4779	4723	**4694**	4715.7
20x20	flcmax_20_20_9	3902	4527	4944	4922	**4720**	4775.4
	flcmax_20_20_2	3881	4499	4886	4847	**4731**	4781.3
	flcmax_20_20_10	3823	4361	4717	4715	**4554**	4607.6
	flcmax_30_15_3	4020	4568	5226	5210	**4927**	5032.2
	flcmax_30_15_4	4080	4649	5304	5284	**5033**	5092.4
30x15	flcmax_30_15_9	4022	4568	5079	5075	**4912**	4968.6
	flcmax_30_15_8	4490	4836	5605	5593	**5220**	5320.2
	flcmax_30_15_6	4184	4761	5147	5149	**5097**	5158.5
	flcmax_30_20_3	4806	5376	6183	5987	**5794**	5846.9
	flcmax_30_20_1	4772	5698	6037	**5989**	6179	6221.8
30x20	flcmax_30_20_6	5004	5752	6241	6195	**6039**	6133.9
	flcmax_30_20_10	4899	5464	6095	5923	**5888**	5967.7
	flcmax_30_20_2	4757	5369	**5822**	5840	5842	5886.1
	flcmax_40_15_5	5560	5958	6986	6972	**6521**	6594.1
	flcmax_40_15_9	5119	5692	6351	6310	**6244**	6303.3
40x15	flcmax_40_15_2	5290	5877	6506	6532	**6302**	6395.6
	flcmax_40_15_10	5596	5896	6845	6712	**6413**	6445.2
	flcmax_40_15_8	5576	6054	6783	6771	**6526**	6611.7
	flcmax_40_20_3	5693	6508	7154	**7132**	7208	7274.5
	flcmax_40_20_9	5998	6676	7528	7496	**7388**	7484.7
40x20	flcmax_40_20_6	5990	6798	7469	7476	**7455**	7553.1
	flcmax_40_20_7	6170	6766	7608	7588	**7405**	7473.5
	flcmax_40_20_5	6011	6508	7219	**7217**	7326	7399.4
	flcmax_50_15_6	6290	6836	7673	7631	**7559**	7606.8
	flcmax_50_15_5	6355	6672	7679	7496	**7317**	7368.4
50x15	flcmax_50_15_1	6198	6580	7416	7402	**7205**	7303.8
	flcmax_50_15_8	6312	6799	7548	7558	**7348**	7468.7
	flcmax_50_15_2	6531	6954	7750	7712	**7547**	7644.8

	flcmax_50_20_2	6740	7682	8838	8836	**8436**	8684.4
	flcmax_50_20_1	6736	7313	8539	8521	**8064**	8189.7
50x20	flcmax_50_20_7	6756	7622	8417	8425	**8370**	8526
	flcmax_50_20_8	6897	7480	8590	8536	**8430**	8509.2
	flcmax_50_20_4	6830	7726	**8493**	8502	8538	8625.1

Table 5. Aggregated results on Taillard and Demirkol benchmarks of size $n \times m$. Yagmahan and Yenisey (2010) is an ant colony system, Lin and Ying (2009) is a hybrid simulated annealing – tabu search; Rajendran (1995) and Ravindran et al. (2005) are two native heuristics, Demirkol et al. (1998) is a shifting bottleneck heuristic; Ying and Lin (2007) is an ant colony system. Native-NPFS ACO is the ant colony system proposed in this paper. In bold the best performing native NPFS approach.

Approach		**Non-native Approach**		**Native Approach**				
Algorithm → Aggregate Instance ↓		Yagmahan and Yenisey (2010)	Lin and Ying, 2009	Rajendran (1995)	Ravindran et al. (2005)	Demirkol et al. (1998)	Ying and Lin (2007)	native-NPFS ACO
20×5	Taillard	6.49	-	8.50	5.71	-	-	**2.44**
20×10	Taillard	11.42	-	13.12	7.67	-	-	**6.94**
20×15	Taillard	11.06	-	7.80	6.98	-	-	**5.87**
20×15	Demirkol	-	0.0	-	-	10.04	8.94	**4.43**
20×20	Demirkol	-	0.0	-	-	8.05	7.51	**5.11**
30×15	Demirkol	-	0.0	-	-	12.74	12.53	**7.73**
30×20	Demirkol	-	0.0	-	-	9.83	8.23	**7.53**
40×15	Demirkol	-	0.0	-	-	13.55	12.96	**8.58**
40×20	Demirkol	-	0.0	-	-	11.19	10.98	**10.60**
50×15	Demirkol	-	0.0	-	-	12.48	11.70	**9.26**
50×20	Demirkol	-	0.0	-	-	13.36	13.21	**10.62**

7.3. Results and discussion

Based on the results presented in **Table 3** to **Table 5** using expression (17), the proposed ACO performs better than others native approaches examined for both Taillard and Demirkol datasets.

Table 3 shows that only few benchmarks are left to the competitors approaches: three to the heuristic of Ravindran et al. (2005) and one to the heuristic of Rajendran (1995). In the other 24 of 28 Taillard sets, it is incumbent.

The $\%gap_{average}$ calculated as in (18) for all sets ranges within 2.2%.

Table 4 shows that the proposed native approach performs better in 35 Demirkol benchmark problems leaving only five to the competitors approaches, particularly three sets to Ying and Lin (2007) and two sets to Demirkol et al. (1998).

Here, the $\%gap_{average}$ ranges like Taillard benchmark except for one problem that exceeds 2.9%.

Table 4 also shows the difference of performance from the non-native approaches considered. Due to a lack of tight upper bounds similar to those evaluated by Vaessens (1996), the upper bound of Demirkol benchmark coincide with the best known solution. Thus, Yagmahan and Yenisey (2010) is not close to the upper bound whilst Lin and Ying (2009) is right the current best-known solution.

Also, native-NPFS ACO performs better in 75% of Taillard benchmark problems than the current best non-native approach, the ACO with pheromone trails initialized by the NEH heuristic, from Yagmahan and Yenisey (2010), making the proposed algorithm a very competitive ant colony system. The non-native hybrid SA – tabu search from Lin and Ying (2009) performs better than other native approaches. Considering that the hybrid metaheuristic of Zobolas et al. (2009) is the state-of-art permutation flowshop (PFS) approach, hybridization is more performing than the individual components. This was also confirmed by Rossi and Boschi (2009) by a Kruskall–Wallis H test statistics on a hybrid ACO – genetic algorithm compared to the two individual components.

However, hybridization is intrinsically opposite of native approach because it is the result of two or more native approaches combined.
Possible reasons for improvements over existing native ACO approaches are inferred from the analysis of **Table 1**, from self complied *parameterized non-delay* candidate list and *freezing function*, and *stability condition* and from other parameters, closer to Yagmahan and Yenisey (2010), which is non-native, than to Ying and Lin (2007).
Table 5 shows the good performance over a wide range of problems has defined by the two benchmarks in **Table 2** in terms of size (# operations) and configuration (*n* to *m* ratio). Individual and aggregated results on all Taillard sets, not listed in **Table 4** because not available from other authors, are available from Lanzetta (2012), where benchmarks are also mirrored, for future benchmarking.

8. Conclusion

Among the main merits of current work is modeling the scheduling of a flow line with buffers as non permutation flowshop, which has also been characterized by a MILP. Alternative approaches and implications have been reviewed in order to map existing metaheuristics approaches to the examined manufacturing problem. The digraph approach described in the paper has the additional merit of being a powerful design tool for native non-permutation algorithms. ACO has been shown to fulfill such design requirements by the pheromone mechanism. The proposed ACO is natively non-permutation as opposed to other authors who apply a local search to permutation solutions. The few existing approaches have been compared using well-known benchmarks. This proposed approach shows the best performance in non-permutation flowshop configuration, particularly on larger instances (available as additional material) and is very close to state-of-the-art metaheuristics, also for non-native. Computation experiments have shown promising performance of such general-purpose optimization tool, regardless of the problem complexity increase in the examined range, from 100 to 10,000 operations and job to machine ratios from 1 to 100.

References

Alaykyran K., Engin O. & Doyen, A. (2007). Using ant colony optimization to solve hybrid flowshop scheduling problems. *International Journal of Advanced Manufacturing Technology*, 35(5–6), 541–550.

Aloulou, M.A. & Artigues, C. (2010). Flexible solutions in disjunctive scheduling: General formulation and study of the flow-shop case. *Computers & Operations Research*, 37(5), 890-898.

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.

Colledani, M. & Tolio, T. (2005). A Decomposition Method to Support the Configuration / Reconfiguration of Production Systems, *CIRP Annals - Manufacturing Technology*, 54(1), 441-444.

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 colonies for the travelling salesman problem. *Biosystems*, 43(2), 73-81.

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. *Operation 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.

Khalili, M., Tavakkoli-Moghaddam, R. (2012). A multi-objective electromagnetism algorithm for a bi-objective flowshop scheduling problem. *Journal of Manufacturing Systems*, 31(2), 232-239.

Koulamas, C. (1998). A new constructive heuristic for the flowshop scheduling problem. *European Journal of Operational Research*, 105, 66–71.

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

Lanzetta, M. (2012) http://www.ing.unipi.it/lanzetta/aco, last acc. 2012.

Leisten, R. (1990). Flowshop sequencing problems with limited buffer storage. *International Journal of Production Research*, 28(11), 2085-2100.

Li, B., Wu S., Yang, J., Zhou, Y. & Du M. (2012). A three-fold approach for job shop problems: A divide-and-integrate strategy with immune algorithm. *Journal of Manufacturing Systems*, 31(2), 195-203.

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

Liao, L.-M. & Huang, C.-J. (2010). Tabu search for non-permutation flowshop scheduling problem with minimizing total tardiness. *Applied Mathematics and Computation*, 217, 557–567.

Lin, S.W. & Ying, K.C. (2008). Applying a hybrid simulated annealing and tabu search approach to non-permutation flowshop scheduling problems. *International Journal of Production Research*, 46, 1–14.

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.

Mehravaran, Y. & Logendran, R. (2012). Non-permutation flowshop scheduling in a supply chain with sequence-dependent setup times. *International Journal of Production Economics*, 135(2), 953-963.

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.

Park, Y. B., Pegden, C. D., Enscore, E.E. (1984). A survey and evaluation of static flowshop scheduling heuristics. *International Journal of Production Research*, 22, 127-141.

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. (2003). Performance enhancement by using non-permutation schedules in flowline-based manufacturing systems. *Computers and Industrial Engineering*, 44, 133–157.

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.

Rajendran, C. (1995). Heuristics for scheduling in flowshop with multiple objectives. *European Journal of Operational Research*, 82, 540–555.

Ravindran, D., Noorul Haq, A., Selvakuar, S.J., & Sivaraman, R. (2005). Flow shop scheduling with multiple objective of minimizing makespan and total flow time. *International Journal of Advanced Manufacturing Technology*, 25, 1007–1012.

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 & Computer Integrated Manufacturing*, 23, 503–516.

Rossi, A. & Boschi E. (2009). A hybrid heuristic to solve the parallel machines job-shop scheduling problem. *Advances in Engineering Software*, 40(2), 118-127.

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*, 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.

Ruiz, R., Maroto, C. & Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, 34(5), 461-476.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1), 65-74.

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

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

Tasgetiren, M.F., Liang Y.-C., Sevkli, M., Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, 177(3), 1930-1947.

Vaessens, R. J. M., http://www.mathematik.uni-osnabrueck.de/research/OR/fsbuffer/taillard2.txt, 1996, last acc. 2012.

Weng, M.X. (2000). Scheduling flow-shops with limited buffer spaces. In J. A. Joines, R. R. Barton, K. Kang, & P. A. Fishwick (Eds.), *Proceedings of the 2000 Winter Simulation Conference*. 1359–1363.

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(2), 1361-1368.

Ying, K.-C. & Liao, C.-J. (2004). An ant colony system for permutation flow-shop sequencing. *Computers & Operations Research*, 31, 791–801.

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. (2008). Solving non-permutation flowshop scheduling problems by an effective iterated greedy heuristic. *International Journal of Advanced Manufacturing Technology*, 38, 348–354.

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.

Zobolas, G.I.. Tarantilis, C.D., Ioannou, G. (2009). Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research*, 36(4), 1249-1267.