Int J Adv Manuf Technol (2017) 89:765–777
DOI 10.1007/s00170-016-9093-3

ORIGINAL ARTICLE

Scheduling flexible flow shop with recirculation and machine sequence-dependent processing times: formulation and solution procedures

Hannu Ahonen[1] · **Arlindo Gomes de Alvarenga**[1]

Received: 2 March 2016 / Accepted: 26 June 2016 / Published online: 11 July 2016

Abstract The focus of this paper is on the treatment of a reentrant and flexible flow shop problem in which the processing times of the jobs at some stage may depend on the decisions made for the jobs at stages before and after the current stage, that is, they may depend on the machine sequence the jobs take in the processing flow. The problem was encountered in a cutting stock application embedded in the context of a virtual organisation. A mathematical model capturing the issues of reentrancy and machine sequence dependency is given. Solution procedures using a mixed-integer programming (MIP) solver and two metaheuristics, simulated annealing and tabu search are presented. The feasibility of the approach is established by computational tests with 30 randomly generated problem instances. The optimal results were obtained for all instances up to ten clients and five service providers and one instance with 15 clients and five service providers. The rest of the results were within the limits provided by the MIP solver.

Keywords Production · Flexible reentrant flow shop · Machine sequence dependency · MIP · Simulated annealing · Tabu search

1 Introduction

The production process implemented in the form of a conventional flow shop consists of a sequence of operations that have to be executed on every job in the same order. This means that the jobs are processed at successive stages. Usually, only one machine is available at each stage. The buffers between the stages may have an unlimited capacity or they may accept only a limited number of waiting jobs.

A flexible (or hybrid) flow shop may have more than one machine (server) available for each stage, and the machines at each stage may be identical, uniform or unrelated. As showed in Gupta [19], the flowshop problem with multiple processors (FSMP) at each stage and with only two stages is NP-Hard, even when one of the two stages contains a single machine. Since the FSMP problem can be considered as a specific case of the flexible flowshop, it can be concluded that the latter problem is also NP-Hard, Ruiz and Maroto [36]. In this work, we assume that the machines are unrelated and that the buffers between stages have unlimited capacities.

The recirculation of a job means that the job may return to a machine, which already processed one of its previous operations. In the flexible flow shop context, a job need not return to the same machine but to one of the machines of a previous stage. This means from the machine viewpoint that some machines are able to process operations of different stages. The flow shop problem with recirculation is often designated as a reentrant flow shop problem.

Related to the sequence-dependent processing times, it is assumed that the processing time of a job on a machine at a stage may depend on the machines selected for that job at the previous and next stages. This assumption, which may sound less natural in the manufacturing context, was motivated by the treatment of transportation stages in the cutting stock application considered by Ahonen et al. [2] and will be described in Section 3. Now, the viewpoint is that of a job, since it is considered what happened to the job before the current stage and what will happen after it. This

✉ Hannu Ahonen
hannu@inf.ufes.br

[1] Departamento de Informática, Universidade Federal do Espírito Santo, 29060-900 Vitória, ES, Brazil

is different from the machine viewpoint, as in the case of setup times, in which the completion time of a job depends on the previous job on the machine. For our knowledge, this kind of dependency has not yet been treated in the literature.

In Ahonen et al. [2], a cutting stock application was embedded in a virtual organisation context so that multiple machines (servers) were selected for each processing stage, that is, for determining the layout of the pieces on the plate, for supplying the material to be cut, for transporting the material to the cutters, for executing the cutting and for transporting the final product to a client. As will be seen in Section 3, the application gives rise to a reentrant flexible flow shop problem with machine sequence-dependent processing times.

The mathematical model and computational experiments in Ahonen et al. [2] were restricted to the basic case in which recirculation and sequence dependencies were not considered. In this paper, we remove these limitations by extending the basic model to contemplate recirculation and sequence dependencies as well.

The paper is organised as follows. A literature review of the related research is provided in Section 2. Section 3 contains a short description of the scheduling problem related to the cutting stock application considered. A mathematical model for the problem is presented in Section 4. The algorithms implemented in this work are explained in Section 5. The results of the computational experiments are given in Section 6. Finally, the conclusions are drawn in Section 7.

2 Related research

In this section, a review of the literature related to the problems involved is given. Two subsections are presented, one treating the flexible or hybrid flow shop scheduling problem and the other related to the recirculation of jobs.

2.1 The flexible/hybrid flow shop scheduling problem

A number of research has been developed for different forms of the flow shop problem. Weng et al. [41] considered the dynamic hybrid flow shop. They developed several routing strategies which can with the help of dispatching rules realise a just-in-time completion of jobs arriving dynamically in the production environment. A fuzzy approach combining hybrid genetic algorithm and data mining was proposed by Zare and Fakhrzad [42] for the flexible flow shop problem.

Ruiz et al. [35] worked with a hybrid flow shop scheduling problem considering several restrictions and characteristics: release dates, unrelated parallel machines at each stage, sequence-dependent setup times, machine eligibility, time lags on operations and precedence constraints among jobs.

Lee [25] analysed two-stage hybrid flowshops with dynamic order arrivals and presented methods for estimating the flowtimes of orders, where a flowtime was defined as the interval between the arrival and the completion of an order. Pan et al. [32] proposed an effective discrete artificial bee colony algorithm based on a hybrid representation of the solutions and a combination of forward/backward decoding methods for solving the hybrid flow shop scheduling problem with the objective of minimising the makespan. The numerical results showed that the proposed algorithm performs better than the best ones encountered in the literature. A hybrid variable neighbourhood search algorithm for solving the hybrid flow shop scheduling problem was proposed in [26].

Bozorgirad and Logendran [7] considered a group scheduling problem in a hybrid flow shop environment and developed a mixed-integer linear programming model to the problem. They proposed a two-level tabu search procedure, which firstly identifies the best sequence of groups and then searches for the best sequence of jobs within each group. A specific hybrid flow shop scheduling problem inspired by a real microelectronic manufacturing environment was treated by Costa et al. [13] with a development of an optimisation approach based on a metaheuristic algorithm powered by a dual search mechanism based on two different problem encodings. Wang and Lin [40] analysed a two-stage no-wait hybrid flow shop having a single machine at stage one and multiple parallel identical machines at stage two and with the objective of makespan minimisation. A genetic algorithm to solve the problem was developed.

Sequence-dependent setup times in a hybrid flow shop were treated in Naderi et al. [31] with a novel simulated annealing algorithm to minimise makespan and maximum tardiness, and in Naderi et al. [30], where two algorithms, a dynamic dispatching rule heuristic and an iterated local search metaheuristic, were presented. Joo et al. [21] considered a three-stage dynamic flexible flow shop problem, in which jobs of multiple types arrive dynamically over time. A scheduling algorithm based on the general dispatching rule-based scheduling procedure was presented to solve the problem. Allahverdi et al. [4] give an extensive survey of scheduling problems with setup times.

Surveys on the flexible (hybrid) flow shop problem, in general, can be found in Linn and Zhang [28], Wang [39], Kis and Pesch [23], Quadt and Kuhn [33], Ruiz and Vázquez-Rodríguez [37] and Ribas et al. [34].

2.2 Recirculation of jobs

Bertel and Billaut [5] proposed a genetic algorithm for solving a hybrid flow shop problem, in which recirculation, release times and due dates of the jobs were considered and

the objective was to minimise the weighted number of tardy jobs.

A multi-objective approach was applied to the reentrant hybrid flow shop scheduling problem in Dugardin et al. [16]. Several heuristic procedures for the *m*-machine reentrant flow shop problem with makespan minimisation were presented in Choi and Kim [12]. Jing et al. [20] considered reentrant flow shop scheduling problems with the objective of minimising total completion time and proposed a heuristic algorithm, in which an effective k-insertion technique was introduced as an improvement strategy at each iteration.

Kang et al. [22] developed a tabu search procedure for the reentrant flow shop problem with sequence-dependent setup times to minimise the total weighted tardiness in the application context of a semiconductor manufacture process. A real-time scheduling procedure with a case study on a thin film transistor-liquid crystal display (TFT-LCD) manufacturing was presented in Choi et al. [11]. A brief review of the scheduling techniques developed for this application can be found in Gupta and Sivakumar [18].

A review on the current practice of what have been done for the integration of dispatching rules and evolutionary algorithms on semiconductor manufacturing scheduling was given in [10]. A hybrid tabu search and a hybrid genetic algorithm for the reentrant flow shop scheduling problem were proposed by Chen et al. [8, 9], respectively.

A unified treatment of several managerial issues for a family of reentrant flow lines was considered in Bispo and Tayur [6]. These issues include capacity allocation, inventory management and production control. Demirkol and Uzsoy [14] proposed a decomposition method for minimising the maximum lateness in the context of the reentrant flow shop problem with sequence-dependent setup times.

The problem of minimising the cycle time or the makespan in *m*-machine reentrant robotic cells was considered by Steiner and Xue [38]. It was shown in the two-machine case that both the cycle time and the makespan can be minimised in polynomial time.

A general survey for different reentrant scheduling environments was presented by Middendorf and Timkovsky [29]. Lin and Lee [27] gave a comprehensive review of the reentrant scheduling problem. The authors classified the reentrant problems and analysed the optimisation methods for the reentrant manufacturing scheduling.

3 A reentrant flow shop with machine sequence-dependent processing times

The questions of reentrant operations and machine sequence-dependent processing times were encountered in a cutting stock production process in the context of a virtual organisation as presented by Ahonen et al. [3] and further analysed by Ahonen et al. [2]. The flexible production process consists of several stages with multiple servers at each stage. There are stages of automated layout design, supply of the material to be cut and execution of the final cutting. There is also a need of transportation at two stages of the process, firstly, from a supplier to a cutter and, secondly, from a cutter to the client. These two cases of transportation can be interpreted as stages of the resulting flow shop problem, as shown in Fig. 1.

Now, since it is assumed that the same transporters can take care of transportation tasks in both cases, the concept of reentrant operations is needed.

Due to possibly different geographical locations, the travels from suppliers to cutters may have different durations. The same applies to the travels from cutters to clients. Thus, the transport times of a task depend on the origin and destination of the travel leading to the inclusion of sequence-dependent processing times in the extended flow shop model.

4 Mathematical model

The mathematical model for the generic flexible flow shop discussed in this article is described in the following four subsections. The first subsection gives definitions needed in the later subsections. The basic model for a flexible flow shop is described in the second subsection, while the extensions for problems with sequence-dependent processing times and recirculation are topics of the last two subsections.

4.1 Definitions

- Indices
  - i: stage
  - j: job
  - k: server

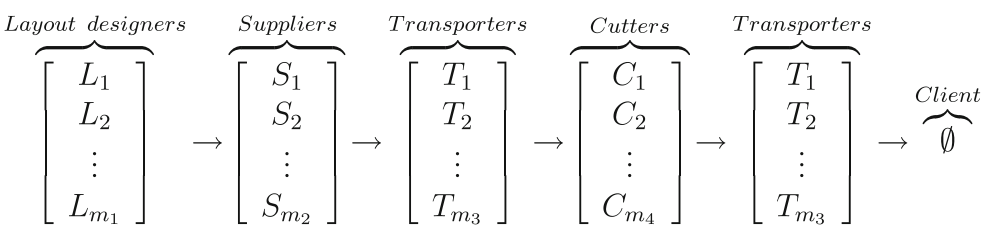

Fig. 1 Processing stages of the production process

– Parametres
  - s: number of stages S^i, $i = 1, 2, \ldots, s$
  - n: number of jobs
  - $\left(O_j^1, O_j^2, \ldots, O_j^s\right)$: operations of job j, $j = 1, 2, \ldots, n$
  - m_i: number of servers R_k^i at each stage S^i, $i = 1, 2, \ldots, s$; $k = 1, 2, \ldots, m_i$; $m_s = 0$ (the last state is fictitious)
  - p_{ijk}: processing time of operation O_j^i by server k
  - M: an integer larger than or equal to an upper bound of the makespan of any schedule

The stages with respective servers are illustrated in Fig. 2.

From now on, we identify a stage S^i with a vector, the elements of which are the servers of the stage, that is, $S^i = \left(R_1^i, R_2^i, \ldots, R_{m_i}^i\right)$ for $i = 1, 2, \ldots, s$.

– Decision variables:
  - t_{ij}: time instant at which operation O_j^i is started.
  - x_{ijk}: is equal to 1, if operation O_j^i is processed by server k, and to 0, otherwise.
  - $y_{ijj'}$: is equal to 1, if operation O_j^i is processed by the same server before job j', $j < j'$, and to 0, otherwise.

– The objective function C_{max} to be minimised: the makespan of the jobs (i.e. the latest task completion time at the last stage).

4.2 The basic model

The values of parametres p_{ijk} express the processing times related to the service providers. The number of jobs, stages and servers at each stage takes some fixed known values. Thus, the decision variables of the problem consist of start times t_{ij}, server selections x_{ijk} and precedences $y_{ijj'}$. An integer linear programming model of the problem in its basic form can be given as follows:

$$\text{minimise} \quad C_{max} \tag{1}$$

subject to the following constraints:

$$C_{max} \geq t_{sj} \quad \text{for} \quad j = 1, 2, \ldots, n. \tag{2}$$

$$\sum_{k=1}^{m_i} x_{ijk} = 1 \quad \text{for} \quad i = 1, 2, \ldots, s-1; \quad j = 1, 2, \ldots, n. \tag{3}$$

$$t_{i+1,j} \geq t_{i,j} + \sum_{k=1}^{m_i} p_{ijk} x_{ijk} \quad \text{for} \quad i = 1, 2, \ldots, s-1; \quad j = 1, 2, \ldots, n. \tag{4}$$

$$t_{ij'} \geq t_{ij} + p_{ijk}x_{ijk} - M(1-y_{ijj'}) - M(1-x_{ijk}) - M(1-x_{ij'k}) \quad \text{for} \quad i = 1, 2, \ldots, s-1; \quad j = 1, 2, \ldots, n-1; \quad j' = j+1, j+2, \ldots, n; \quad k = 1, 2, \ldots, m_i. \tag{5}$$

$$t_{ij} \geq t_{ij'} + p_{ij'k}x_{ij'k} - My_{ijj'} - M(1-x_{ijk}) - M(1-x_{ij'k}) \quad \text{for} \quad i = 1, 2, \ldots, s-1; \quad j = 1, 2, \ldots, n-1; \quad j' = j+1, j+2, \ldots, n; \quad k = 1, 2, \ldots, m_i. \tag{6}$$

$$x_{ijk} \in \{0, 1\} \quad \text{for} \quad i = 1, 2, \ldots, s-1; \quad j = 1, 2, \ldots, n; \quad k = 1, 2, \ldots m_i. \tag{7}$$

$$y_{ijj'} \in \{0, 1\} \quad \text{for} \quad i = 1, 2, \ldots, s-1; \quad j = 1, 2, \ldots, n-1; \quad j' = j+1, 2, \ldots, n. \tag{8}$$

The inequalities (2) together with the minimisation objective force the value of C_{max} to be equal to the start time of the last job reaching the fictitious stage. The constraints (3) stipulate that exactly one server has to be selected for each operation, while the constraints (4) state that an operation of a job can be started only after finishing the operation of that job at the previous stage. Processing two operations on a server at the same time is prevented by the restrictions (5) and (6).

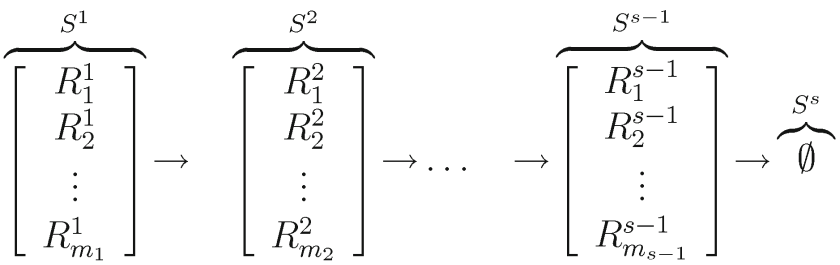

Fig. 2 Stages of a flexible flow shop

4.3 Sequence-dependent processing times

For the first extension, we assume that the system includes special servers, say, transporters, with processing times that depend not only on the jobs to be transported but also on their origins and destinations. A stage having transporters as its servers is called a transporter stage. The transportation of a job is defined as one of the job's operations. We assume that if there are more than one transporter stage, the transporters are distinct. The case of related transporters in distinct stages will be the subject of the next subsection. We identify the origin and destination of a transport with servers that process the job at stages immediately before and after the transporter stage, respectively.

We define the following now:

- $\mathscr{S}^T$: the set of transporter stages
- $p_{ijk'kk''}$: transportation time from server $R^{i-1}_{k'}$ to server $R^{i+1}_{k''}$ needed by transporter R^i_k to complete operation O^i_j

A new set of decision variables is needed:

- $x_{ijk'kk''}$: is equal to 1, if operation O^i_j is a transport operation processed by transporter R^i_k with origin $R^{i-1}_{k'}$ and destination $R^{i+1}_{k''}$, and equal to 0, otherwise.

To shorten the presentation below, we define the following for a transporter stage:

$$x_{ijk} = \sum_{k'=1}^{m_{i-1}} \sum_{k''=1}^{m_{i+1}} x_{ijk'kk''}. \tag{9}$$

Now, the value $x_{ijk} = 1$ indicates that transporter R^i_k processes the transportation operation O^i_j from some origin to some destination. This is in concordance with the definition of x_{ijk} given in the basic model, and the constraints (3) take part in the current model, too.

The extension to the problem with sequence-dependent processing times is now obtained from the basic model by maintaining the constraints (4), (5) and (6) for the stages not being transporter stages ($i \notin \mathscr{S}^T$) and by adding the following expressions:

$$t_{i+1,j} \geq t_{i,j} + \sum_{k'=1}^{m_{i-1}} \sum_{k=1}^{m_i} \sum_{k''=1}^{m_{i+1}} p_{ijk'kk''} x_{ijk'kk''} \quad \text{for} \quad i \in \mathscr{S}^T; \quad j = 1, 2, \ldots, n. \tag{10}$$

$$t_{ij'} \geq t_{ij} + \sum_{k'=1}^{m_{i-1}} \sum_{k''=1}^{m_{i+1}} p_{ijk'kk''} x_{ijk'kk''} - M(1 - y_{ijj'}) - M(1 - x_{ijk}) - M(1 - x_{ij'k}) \quad \text{for} \quad i \in \mathscr{S}^T; \quad j = 1, 2, \ldots, n-1; \quad j' = j+1, j+2, \ldots, n; \quad k = 1, 2, \ldots, m_i. \tag{11}$$

$$t_{ij} \geq t_{ij'} + \sum_{k'=1}^{m_{i-1}} \sum_{k''=1}^{m_{i+1}} p_{ijk'kk''} x_{ijk'kk''} - M y_{ijj'} - M(1 - x_{ijk}) - M(1 - x_{ij'k}) \quad \text{for} \quad i \in \mathscr{S}^T; \quad j = 1, 2, \ldots, n-1; \quad j' = j+1, j+2, \ldots, n; \quad k = 1, 2, \ldots, m_i. \tag{12}$$

$$x_{ijk'kk''} \in \{0, 1\} \quad \text{for} \quad i \in \mathscr{S}^T; \quad j = 1, 2, \ldots, n; \quad k' = 1, 2, \ldots m_{i-1}; \quad k = 1, 2, \ldots m_i; \quad k'' = 1, 2, \ldots m_{i+1}. \tag{13}$$

The interpretation of the constraints in (10), (11) and (12) remains the same as the one given to constraints (4), (5) and (6).

It is not difficult to see that the model can be extended to include dependencies with longer sequences by modifying the sum terms in it.

4.4 Recirculation

If two stages share some of their servers, this means that a server may attend operations of jobs belonging to different stages. This recirculation may happen with operations of the same job, with operations of different jobs, with an operation of an earlier stage first, or vice versa.

The central issue in the case of recirculation is to control that there is no overlap in processing the operations. Again, a new set of variables is needed:

- $y_{iji'j'}$: is equal to 1, if operation O^i_j is processed before operation $O^{i'}_{j'}$ and equal to zero, otherwise.

In addition, let us denote by $\mathscr{C}$ the set of all pairs (i, i') for which there is a server $R \in S^i \cap S^{i'}$, that is, there are indices k and $\bar{k}$ such that $R = R^i_k = R^{i'}_{\bar{k}}$.

The desired control is now achieved with a modification of the constraints (5) and (6) in the basic model as follows:

$$t_{i'j'} \geq t_{ij} + p_{ijk}x_{ijk} - M(1 - y_{iji'j'}) - M(1 - x_{ijk}) - M(1 - x_{i'j'\bar{k}}) \quad \text{for } (i, i') \in \mathscr{C}; \quad j = 1, 2, \ldots, n-1; \quad j' = j+1, j+2, \ldots, n \quad \text{and for } (k, \bar{k}) \text{ with } R^i_k = R^{i'}_{\bar{k}}. \tag{14}$$

$$t_{ij} \geq t_{i'j'} + p_{i'j'k}x_{i'j'k} - My_{iji'j'} - M(1 - x_{ijk}) - M(1 - x_{i'j'\overline{k}})$$
$$\text{for} \quad (i, i') \in \mathscr{C}; \quad j = 1, 2, \ldots, n-1;$$
$$j' = j+1, j+2, \ldots, n$$
$$\text{and for } (k, \overline{k}) \text{ with } R^i_k = R^{i'}_{\overline{k}}. \tag{15}$$

Clearly, if the processing times are sequence-dependent, the modifications of constraints (11) and (12) are analogous.

In addition, if for some pair $(i, i') \in \mathscr{C}$, $S^i = S^{i'}$, then we can take $\overline{k} = k$ in constraints (14) and (15). This is the special case treated in Bertel and Billaut [5].

5 SA and TS algorithms

As indicated by the computational experiments described in the following section, the mixed-integer programming (MIP) solver was not able to solve the largest problem instances up to optimality. For this, two metaheuristics, simulated annealing (SA) [24] and tabu search (TS) [17], were implemented. It is thus possible to have more convincement on the quality of the solutions in the situation in which comparisons with results in the literature are not possible. Our objective is neither to propose novel techniques nor to conclude, which of the algorithms has better performance, but to illustrate, how it is possible to apply a quantitative analysis to the largest instances, too. Thus, the SA and TS offer a natural choice being well-known and reliable metaheuristics, which have been used with success for a wide variety of optimisation problems.

Both algorithms use the same neighbourhood structure for generating new solutions. The structure is based on two types of swap "moves": (1) exchange of processing order of two jobs on a given server and (2) reselection of a server for a given job.

Each solution defines an allocation of the jobs to the servers and their ordering on them. The cost calculation uses this information by constructing the associated schedule with the inclusion of the details of recirculation and server-dependent processing times.

A local search algorithm was implemented and called by both algorithms. It simply applies all moves to the current solution, and if a better solution is found, it is nominated to the new current solution. If, after the application of all moves, a better solution has been found, the loop over the moves is repeated, otherwise the algorithm stops.

The two metaheuristics and the local search algorithm are adaptations to the current neighbourhood structure of the algorithms described in [1].

5.1 SA algorithm

The following 'basicSA' procedure is called repeatedly by the implementation of the SA algorithm.

Procedure basicSA

```
   input  : Initial solution S_0, array Swaps, doubles T_0,
            TempFactor, FinalTemp, integer
            NInnerLoopIters
   output: Solution S
 1 S ← S_0, T ← T_0;
 2 while T > FinalTemp do
 3     iIter ← 0;
 4     while iIter < NInnerLoopIters do
 5         iRandom ← floor(random(0,1) * Swaps.length);
 6         S' ← applySwap(Swaps[iRandom], S);
 7         Δ ← c(S') − c(S);
 8         prob ← min(1, e^{−Δ/T});
 9         if random(0,1) ≤ prob then
10             S ← S';
11         end
12         iIter ← iIter + 1;
13     end
14     T ← TempFactor * T;
15 end
```

The main algorithm, 'Algorithm 1', firstly determines the initial temperature T_0 by calling the procedure 'reversedSA', which is essentially the same as procedure 'basicSA' with a different stopping criterion (at line 2) and with the temperature update factor $ReversedTempFactor$ greater than one, so that the temperature is increased during the outer loop iterations. The execution of the outer loop stops when the acceptance rate in the inner loop reaches 100 %.

Algorithm 1 Simulated annealing with restarts

```
   input  : Initial solution S_0, integer NRestarts,
            double InitialReversedTemp
   output: Best solution SBest
 1 Create array Swaps;
 2 SBest ← S_0, minValue ← c(S_0);
 3 S' ← localSearch (S_0);
 4 if c(S') < minValue then
 5     SBest ← S', minValue ← c(S');
 6     S_0 ← S';
 7 end
 8 T_0 ← reversedSA(S_0, InitialReversedTemp, ...);
 9 for i = 0 to NRestarts do
10     S ← S_0;
11     S' ← basicSA(S, Swaps, T_0, ...);
12     S' ← localSearch(S');
13     if c(S') < minValue then
14         SBest ← S', minValue ← c(S');
15     end
16     T_0 ← T_0/2;
17 end
```

5.2 TS algorithm

The procedure 'pickNextSwap' is used to update the current TS solution with the help of non-tabu swap moves unless a tabu move would lead to a new overall best solution. The tabu status of a move is determined by its recency, that is, if the move was applied at one of the last $TabuTenure$ iterations, it is declared tabu. A varying $TabuTenure$ length was implemented. If no overall best neighbour can be found, the best non-tabu move is selected with respect to a modified cost defined at line 13 of the procedure. In case of large problems, not all neighbours are explored, but only those with indices in the current index set. The set of all indices is split into $NIndexSets$ subsets designated as $IndexSets$. For example, if $NIndexSets = 3$ and the number of swap moves, $Swaps.length = 12$, the subsets would be $\{1, 4, 7, 10\}$, $\{2, 5, 8, 11\}$ and $\{3, 6, 9, 12\}$. These index sets are alternated during the TS iterations.

Procedure pickNextSwap

```
    input  : Current solution S, arrays IndexSet, Swaps,
             SwapFrequencies and SwapRecencies, integers
             iIter, tenure, minValue
    output: Swap index iResult
 1  SBest ← null;
 2  iBest ← −1;
 3  minValueNow ← minValue;
 4  minModifValueNow ← Integer.MAX_VALUE;
 5  for i in IndexSet do
 6      STest ← applySwap(Swaps[i], S);
 7      if c(STest) < minValueNow then
 8          SBest ← STest;
 9          minValueNow ← c(STest);
10          iBest ← i;
11      else if isNonTabu(SwapRecencies[i], iIter, tenure)
        then
12          f ← SwapFrequencies[i];
13          modifCost ← (1 + f/(1 + f)) ∗ c(STest);
14          if modifCost < minModifValueNow then
15              minModifValueNow ← modifCost;
16              SResult ← STest;
17              iResult ← i;
18      end
19  end
20  if iBest ≥ 0 then
21      iResult ← iBest;
22  end
```

'Algorithm 2' describes our implementation of tabu search. The diversification step (procedure 'diversify') at line 30 consists of alternating application of the least recent or the least frequent swap moves.

Algorithm 2 tabuSearch

```
    input  : Initial solution S0, array IndexSets, double
             TabuFactor, integers MinTabuTenure,
             MaxTabuTenure, DivInterval, NMaxIters,
             NDivSteps
    output: Best solution SBest
 1  Create array Swaps;
 2  for i = 0 to Swaps.length do
 3      SwapFrequencies[i] ← 0;
 4      SwapRecencies[i] ← −MaxTabuTenure;
 5  end
 6  NIndexSets ← array length of IndexSets;
 7  iIndexSet ← −1; tenure ← MaxTabuTenure;
    minValue ← c(S0);
 8  S ← S0;
 9  S′ ← localSearch(S);
10  if c(S′) < minValue then
11      S ← S′; minValue ← c(S′); SBest ← S′;
12  end
13  iIter ← 0; iNotImproved ← 0; useRecency ← TRUE;
14  while iIter < NMaxIters do
15      iIter ← iIter + 1;
16      iIndexSet ← iIndexSet + 1  mod NIndexSets
        IndexSetNow ← IndexSets[iIndexSet];
17      i ← pickNextSwap(S, IndexSetNow, ...);
18      SwapFrequencies[i] ← SwapFrequencies[i] + 1;
19      SwapRecencies[i] ← iIter;
20      S ← applySwap(Swaps[i], S);
21      if c(S) < minValue then
22          minValue ← c(S); SBest ← S;
            iNotImproved ← 0;
23          S′ ← localSearch(S);
24          if c(S′) < minValue then
25              S ← S′; minValue ← c(S′); SBest ← S′;
26          end
27      else
28          iNotImproved ← iNotImproved + 1;
29          if iNotImproved mod DivInterval = 0 then
30              S ← diversify(S, Swaps, useRecency,
                NDivSteps);
31              useRecency ← not useRecency;
32          end
33      end
34      tenure ← round(TabuFactor ∗ tenure);
35      if tenure < MinTabuTenure then
36          tenure ← MaxTabuTenure;
37      end
38  end
39  end
```

6 Computational experiments

The computational experiments related to the cutting stock application consist of tests with ten groups of randomly generated instances, with three instances in each group. The instances within a group have the same problem size designated by the number of clients n and the number of service providers at each stage m (selected to be the same number

for all flow shop stages). The processing times of the service providers were picked as integers from the uniform random distribution at interval [5, 100].

All tests were conducted with a standard personal computer equipped with a 2.83-GHz Intel(R) Core(TM) 2 Quad processor and 4 GB of RAM under the Linux operating system. The algorithms were implemented in Java and tested on JVM 1.7.

The MIP package used was CPLEX 12.5.1.0.

We used a slightly modified objective function, in which the sum of start times of the jobs multiplied a small factor was added to makespan:

$$\text{minimise } C_{max} + \varepsilon \sum_{i,j} t_{ij}.$$

The value of ε was selected so small that the sum of the start times always remains smaller than $0.1 * \varepsilon^{-1}$. Thus, the jobs in the optimal solution are started as early as possible and the additional term has no influence on the minimal makespan value.

Throughout the experiments, the following parametre values were used for SA and TS, respectively:

SA:

$$\begin{aligned} InitialReversedTemp &= 0.1/(m \times n), \\ ReversedTempFactor &= 0.999, \\ ReversedNInnerLoopIters &= m \times n/2, \\ TempFactor &= 1 - 1/(5000m \times n), \\ FinalTemp &= 1/(m \times n), \\ NInnerLoopIters &= m \times n, \\ NRestarts &= 10. \end{aligned}$$

Table 1 MIP minimum makespan results for 30 instances

		Obj.	LB	UB	Gap	Gap (%)	Time
n=5	1	282					1.69 s
m=2	2	269					1.52 s
	3	232					0.40 s
n=5	1	214					0.57 s
m=3	2	184					1.42 s
	3	223					0.21 s
n=7	1	343					3.57 s
m=2	2	274					5.47 s
	3	325					21.85 s
n=7	1	225					8.83 s
m=3	2	197					5.56 s
	3	145					1.40 s
n=10	1	231					108.5 s
m=3	2	255					322.6 s
	3	230					100.0 s
n=10	1	139					30.38 s
m=5	2	129					41.63 s
	3	138					32.49 s
n=15	1	–	192	315	123	39.0	2 h
m=3	2	–	211	315	104	33.0	2 h
	3	–	180	262	82	31.3	2 h
n=15	1	–	138	150	12	8.00	2 h
m=5	2	180					1993 s
	3	–	160	175	15	8.57	2 h
n=20	1	–	150	178	28	15.7	2 h
m=5	2	–	124	186	62	33.3	2 h
	3	–	132	177	45	25.42	2 h
n=20	1	–	112	120	8	6.67	2 h
m=7	2	–	124	126	2	1.59	2 h
	3	–	111	123	12	9.76	2 h

Table 2 SA algorithm—makespan values for 30 instances

		Objective values				No. of evaluations (x1000)			
		Min.	Max.	Avg.	S.D	Min.	Max.	Avg.	S.D
n=5	1	282	282	282.00	0.00	29,096	40,576	34,542	2955
m=2	2	269	269	269.00	0.00	27,038	32,714	30,099	1587
	3	232	232	232.00	0.00	26,202	33,128	29,494	2140
n=5	1	214	214	214.00	0.00	7992	13,092	10,228	1284
m=3	2	184	184	184.00	0.00	11,943	17,268	14,888	1280
	3	223	223	223.00	0.00	13,264	16,226	14,751	999
n=7	1	343	343	343.00	0.00	72,293	87,632	80,751	3599
m=2	2	274	279	274.50	1.53	76,359	85,266	81,344	2736
	3	325	330	325.17	0.91	78,614	94,723	85,764	3946
n=7	1	225	225	225.00	0.00	24,623	30,520	27,948	1467
m=3	2	197	201	197.23	0.90	21,256	26,366	24,581	1281
	3	145	145	145.00	0.00	22,479	26,387	24,521	981
n=10	1	231	235	232.67	1.06	38,815	48,988	44,017	2507
m=3	2	255	259	256.97	0.67	42,084	50,330	45,447	1834
	3	230	230	230.00	0.00	38,643	46,447	43,537	2270
n=10	1	139	142	139.43	0.90	20,357	24,577	22,427	1180
m=5	2	129	131	129.70	0.84	20,588	24,519	22,240	1101
	3	138	143	138.93	1.34	20,438	24,461	22,727	1021
n=15	1	288	299	293.37	3.17	85,742	96,912	91,062	2718
m=3	2	292	300	295.70	2.40	85,748	96,563	90,253	3268
	3	244	250	246.10	1.95	82,722	94,568	89,683	2970
n=15	1	152	160	155.53	2.03	34,763	45,283	40,113	2048
m=5	2	180	181	180.13	0.35	37,916	45,894	43,575	2000
	3	170	184	178.87	3.45	35,892	44,981	41,987	1964
n=20	1	171	185	177.90	2.99	58,651	70,995	66,293	2978
m=5	2	172	183	177.70	2.72	59,058	72,897	66,801	3273
	3	174	186	179.60	3.09	58,306	71,952	66,906	3095
n=20	1	126	142	133.63	3.11	48,158	57,425	54,114	2403
m=7	2	135	144	139.53	2.28	48,518	57,967	53,364	2496
	3	130	140	135.67	2.63	44,994	59,453	53,031	2983

TS:

$$\begin{aligned} NMaxIters &= 100,000, \\ DivInterval &= 10,000, \\ NDivSteps &= 10, \\ MinTabuTenure &= min\{3, \lfloor 0.1m \times n \rfloor\}, \\ MaxTabuTenure &= 2 \times MinTabuTenure, \\ TabuFactor &= 0.995. \end{aligned}$$

The parametre values were determined with the help of preliminary test runs.

As shown in Table 1, optimal makespan values were obtained for all instances up to ten clients and five service providers. The second instance of 15 clients and five service providers was solved optimally, too. In fact, a preliminary test with a longer processing time (5 h) gave the optimal value 149 for the first instance in this group. Generally, the maximum processing time used in the tests was 2 h.

The tests concerning the implementations of SA and TS algorithms were organised as follows: Each one of the 30 instances was solved repeating the test for 30 times. Randomly generated initial solutions were used. Each test was terminated when no improvement took place during 10,000 iterations. The minimum, maximum and average values of the makespan as well as their standard

Table 3 TS algorithm—makespan values for 30 instances

		Objective values				No. of evaluations (x1000)			
		Min.	Max.	Avg.	S.D	Min.	Max.	Avg.	S.D
n=5	1	282	282	282.00	0.00	17,262	17,299	17,279	11
m=2	2	269	269	269.00	0.00	16,995	17,049	17,021	11
	3	232	232	232.00	0.00	17,305	17,340	17,322	9
n=5	1	214	214	214.00	0.00	22,589	22,631	22,609	10
m=3	2	184	184	184.00	0.00	22,277	22,342	22,311	17
	3	223	223	223.00	0.00	22,360	22,401	22,376	11
n=7	1	343	343	343.00	0.00	33,849	33,932	33,893	22
m=2	2	274	279	274.33	1.27	33,019	33,194	33,103	46
	3	325	325	325.00	0.00	33,820	33,874	33,852	15
n=7	1	225	225	225.00	0.00	42,655	42,795	42,733	34
m=3	2	197	197	197.00	0.00	42,577	42,739	42,655	39
	3	145	145	145.00	0.00	42,434	42,508	42,463	20
n=10	1	231	233	232.73	0.69	84,925	85,146	85,045	59
m=3	2	255	257	256.50	0.82	84,133	84,342	84,247	46
	3	230	230	230.00	0.00	84469	84645	84576	46
n=10	1	139	139	139.00	0.00	67,489	67,663	67,583	41
m=5	2	129	130	129.10	0.30	67,669	67,767	67,710	28
	3	138	142	138.43	0.86	67,701	67,796	67,747	22
n=15	1	288	299	293.53	2.93	150,822	151,372	151,119	140
m=3	2	291	303	297.40	2.85	151,209	151,564	151,393	88
	3	244	255	246.43	2.53	151,339	151,872	151,591	128
n=15	1	153	160	155.00	1.49	111,338	111,749	111,560	90
m=5	2	180	182	180.23	0.50	111,889	112,090	111,982	53
	3	173	183	177.93	2.49	112,057	112,268	112,184	53
n=20	1	169	185	174.70	3.47	172,481	172,897	172,722	88
m=5	2	172	182	175.80	3.16	172,955	173,411	173,216	98
	3	172	183	178.00	2.80	172,466	172,812	172,605	86
n=20	1	124	137	129.67	3.17	142,288	142,574	142,428	78
m=7	2	130	141	133.50	2.62	142,313	142,585	142,434	69
	3	124	140	131.87	3.20	142,434	142,730	142,628	71

Table 4 Results of the application of the Mann-Whitney-Wilcoxon test

			Sum-min	*z*
n=7	2	SA	465.0	0.577
m=2	2	TS	435.0	
n=10	1	SA	431.0	
m=3	1	TS	469.0	0.661
	2	SA	573.0	4.282**
	2	TS	327.0	
n=10	2	SA	625.5	3.974**
m=5	2	TS	274.5	
	3	SA	552.5	2.278*
	3	TS	347.5	
n=15	1	SA	448.0	
m=3	1	TS	452.0	0.041
	2	SA	296.0	
	2	TS	604.0	3.247**
	3	SA	430.0	
	3	TS	470.0	0.422
n=15	1	SA	513.5	1.317
m=5	1	TS	386.5	
	2	SA	418.0	
	2	TS	482.0	0.954
	3	SA	559.0	2.261*
	3	TS	341.0	
n=20	1	SA	702.5	5.237**
m=5	1	TS	197.5	
	2	SA	608.0	3.277**
	2	TS	292.0	
	3	SA	576.5	2.623**
	3	TS	323.5	
n=20	1	SA	736.5	5.942**
m=7	1	TS	163.5	
	2	SA	853.0	8.358**
	2	TS	47.0	
	3	SA	744.0	6.097**
	3	TS	156.0	

(*, **) indicate the significancy of the normalized difference *z* at levels 95 % and 99 %, respectively.

deviations were computed. The results are shown in Tables 2 and 3. It can be observed that the results obtained by both algorithms are compatible with those shown in Table 1.

Although the objective of this paper is not to compare the two metaheuristics, Table 4 shows the results of the Mann-Whitney-Wilcoxon test [15] in the case of the instances for which the two metaheuristics produced non-zero standard deviations. The values in the column "Sum-min" consist of the rank sum of the concatenated SA-TS results subtracted by the minimum rank sum, that is, the sum 1 + 2+ ...+ 30 = 465. The results show that among 17 reported instances, only in one of them (n=15, m=3, instance 2) SA gave a significantly better result (at 99 % level). The TS metaheuristic was better twice at 95 % level and eight times at 99 %.

The CPU times are given in Tables 5 and 6. The right hand sides of the tables show the CPU times needed to find the best solution.

Tables 2 and 3 show that the makespan values remained greater than the MIP upper bounds in the case of the first instance with $n = 15$ and $m = 5$ and when $n = 20$ and $m = 7$. For this, we performed additional SA tests, in which new types of restarts were introduced. Each time the final temperature was achieved, the processing was restarted from the current solution with a temperature equal to the final temperature multiplied by ten. These restarts were repeated ten times. The results and the CPU times are presented in Tables 7 and 8. It can be observed that the makespan values are now less than or equal to the upper bounds. The average CPU times are between 18 and 170 min, approximately.

Table 5 SA algorithm—CPU times for 30 instances

		CPU times (ms)				CPU times of best (ms)			
		Min.	Max.	Avg.	S.D	Min.	Max.	Avg.	S.D
n=5	1	46,898	65,669	56,539	4981	1	12,770	2191	3450
m=2	2	42,736	52,021	48,199	2672	3	788	125	159
	3	41,281	54,750	48,060	4013	11	569	126	123
n=5	1	20,143	33,548	26,350	3570	5	1178	302	332
m=3	2	30,842	45,284	39,139	3578	4	14539	2303	4039
	3	34,054	43,161	38,971	2745	4	305	115	89
n=7	1	137,968	166,629	153,691	6715	2186	89,830	18,801	20,667
m=2	2	144,566	165,012	156,090	5945	1002	132,299	46,179	42,040
	3	146,365	175,794	159,955	7537	2626	140,971	43,876	45,755
n=7	1	70,218	87,794	79,892	4544	2000	55,472	21,787	14,926
m=3	2	60,017	76,325	70,126	4241	2378	55,032	15,548	13,936
	3	60,607	72,949	66,889	3229	89	24,141	3982	4852
n=10	1	119,350	150,872	134,638	7766	5935	124,677	52,073	35,985
m=3	2	137,869	164,610	147,894	6794	5999	133,111	41,436	37,304
	3	121,435	151,709	141,429	7743	7078	70,837	30,753	20,940
n=10	1	112,866	145,438	128,687	8759	7178	118,703	41,378	34,159
m=5	2	112,882	138,423	125,655	7267	9122	127,685	53,120	34,088
	3	110,838	140,326	127,392	6758	19,730	127,615	66,965	34,774
n=15	1	348,029	400,887	376,162	12,800	26,591	355,374	217,405	101,934
m=3	2	338,262	387,881	358,979	14,718	26,938	367,483	166,244	107,877
	3	340,586	403,801	369,430	14,213	21,995	355,150	166,961	87,908
n=15	1	241,607	315,872	280,633	16,391	21,951	286,734	142,413	82,237
m=5	2	253,234	314,604	294,311	15,011	819	281,948	107,802	80,726
	3	243,931	304,356	288,662	13,933	1364	297,415	133,638	83,180
n=20	1	495,149	604,085	550,247	25,891	51,035	507,250	257,876	158,949
m=5	2	478,893	581,231	539,434	28,457	49,322	552,428	268,551	140,355
	3	484,167	578,469	540,064	25,439	54,111	559,852	261,537	162,999
n=20	1	554,610	661,257	618,761	28,257	3523	625,450	260,176	175,610
m=7	2	556,958	670,707	619,479	31,908	3606	652,873	283,266	177,613
	3	508,413	697,097	621,675	39,209	4460	644,107	358,190	168,268

Table 6 TS algorithm—CPU times for 30 instances

		CPU times (ms)				CPU times of best (ms)			
		Min.	Max.	Avg.	S.D	Min.	Max.	Avg.	S.D
n=5	1	22,047	23,377	22,415	375	9	1839	691	497
m=2	2	21,386	22,770	21,896	344	4	574	115	144
	3	21,325	23,970	21,812	565	5	179	56	51
n=5	1	34,348	36,314	34,729	365	13	2954	995	887
m=3	2	33,624	36,039	34,427	624	6	6269	1660	1156
	3	33,561	35,447	34,195	578	10	2496	1606	930
n=7	1	52,258	55,476	53,013	638	148	12,572	4541	3069
m=2	2	51,270	53,388	51,820	499	649	45,792	12,241	12,438
	3	52,610	54,461	53,376	490	263	41,993	12,229	12,431
n=7	1	76,025	84,299	78,038	1749	220	54,540	13,437	11,117
m=3	2	77,147	82,128	78,449	1287	987	39,494	10,575	8944
	3	76,839	81,292	78,270	1308	315	14,452	4287	3725
n=10	1	190,159	199,326	192,654	2303	13,380	118,157	45,638	31,689
m=3	2	194,703	203,176	197,213	2165	3878	135,096	35,666	34,274
	3	196,088	203,908	197,290	1532	1160	64,293	21,892	15,899
n=10	1	202,683	212,022	204,832	2418	8075	129,872	60,325	35,453
m=5	2	201,864	214,037	206,450	2550	21,932	195,212	84,424	48,013
	3	199,713	206,482	202,424	1527	5672	200,692	95,398	55,732
n=15	1	488,538	511,019	496,189	5947	27,764	464,123	273,507	123,215
m=3	2	519,171	533,338	523,154	3825	16,031	508,221	226,209	159,550
	3	496,723	502,874	499,740	1857	52,589	493,641	238,376	128,875
n=15	1	467,970	484,900	474,496	4051	53,445	460,893	244,413	108,651
m=5	2	473,526	491,086	479,028	3785	47,574	452,901	253,000	110,105
	3	464,160	487,182	469,913	4904	66,107	470,343	331,076	109,473
n=20	1	969,581	1,055,694	990,975	27,251	155,310	985,382	655,829	221,563
m=5	2	974,175	998,281	984,807	6784	187,799	969,635	616,736	208,714
	3	996,460	1,036,654	1,005,604	8594	284,418	1,005,561	610,805	222,410
n=20	1	945,314	1,006,516	959,948	15,856	364,600	979,586	766,307	160,048
m=7	2	935,668	958,091	942,208	4853	41,195	934,119	600,388	255,646
	3	1,000,694	1,105,721	1,037,029	21,299	252,314	1,055,394	737,109	206,246

Table 7 SA algorithm—additional makespan values for four instances

		Objective values				No. of evaluations (x1000)			
		Min.	Max.	Avg.	S.D	Min.	Max.	Avg.	S.D
n=15	1	150	156	153.73	1.05	179,700	183,048	181,185	794
m=5									
n=20	1	119	124	121.03	1.77	1,206,689	1,267,865	1,239,428	16,569
m=7	2	126	134	130.60	1.61	253,514	258,070	255,658	1214
	3	120	131	126.37	2.33	253,480	258,111	255,418	1250

Table 8 SA algorithm—additional CPU times for four instances

		CPU times (ms)				CPU times of best (ms)			
		Min.	Max.	Avg.	S.D	Min.	Max.	Avg.	S.D
n=15	1	1,129,368	1,158,517	1,138,315	4830	1,129,368	1,158,517	1,138,315	4830
m=5									
n=20	1	9,866,944	10,394,638	10,177,940	125,050	993,487	10,279,847	4,697,472	3,005,375
m=7	2	2,591,987	2,622,561	2,603,879	7736	2,591,987	2,622,561	2,603,879	7736
	3	2,653,717	2,731,814	2,672,518	14,602	2,653,717	2,731,814	2,672,518	14,602

7 Conclusion

The mathematical model developed in this work captures a generalisation of a reentrant flexible flow shop, in which the processing times of the jobs at some stage may depend on the decisions made for the jobs at stages before and after the current stage, that is, they may depend on the machine sequence the jobs take in the processing flow. The problem was encountered in a cutting stock application embedded in the context of a virtual organisation.

The cutting stock problem is encountered in several production contexts, such as cutting steel or glass, or cutting wood for furniture making, in which rectangular pieces are cut from a single rectangular stock plate. We can also consider applications related to irregular cutting or nesting problem class; for example, industries working with leather and textile need to cut pieces using moulds of irregular shape, and industries such as shipyards cut steel plates into irregular shapes to build ships.

The model presented in this paper was applied to the cutting stock application presented in Ahonen et al. [2]. Here, same machines or servers can process tasks belonging to different stages, and the processing times may depend on the machines or servers used at stages before and after the current stage. When compared to the basic flow shop model, the number of decision variables tends to be large. This was observed in the case of instances with number of clients (jobs) $n = 15, 20$ and number of servers $m = 3, 5$ and 7, for which, except two cases of $n = 15$, $m = 5$, no optimal solutions were found with the MIP package used in this work. In this situation, a practical choice would be to use a metaheuristic algorithm, like simulated annealing or tabu search, as illustrated in this work.

It can be considered that the mathematical model and the use of metaheuristics could form a basis for introducing practical tools for system development in the context of flexible flow shops as defined in this paper, or more specifically, in the context of cutting stock applications.

References

1. Ahonen H, Alvarenga AG, Amaral ARS (2014) Simulated annealing and tabu search approaches for the corridor allocation problem. Eur J Oper Res 232:221–33
2. Ahonen H, Alvarenga AG, Provedel A (2009) Selection and scheduling in a virtual organisation environment with a service broker. Comput Indust Eng 57:1353–62
3. Ahonen H, Alvarenga AG, Provedel A, Parada V (2001) A client-broker-server architecture of a virtual enterprise for cutting stock applications. Int J Comput Integr Manuf 14(2):194–205
4. Allahverdi A, Ng CT, Cheng TCE, Kovalyov MY (2008) A survey of scheduling problems with setup times or costs. Eur J Oper Res 187:985–1032
5. Bertel S, Billaut J-C (2004) A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. Eur J Oper Res 159:651–62
6. Bispo CF, Tayur S (2001) Managing simple re-entrant flow lines: theoretical foundation and experimental results. IIE Trans 33:609–23
7. Bozorgirad MA, Logendran R (2013) Bi-criteria group scheduling in hybrid flowshops. Int J Prod Econ 145:599–612
8. Chen J-S, Pan JC-H, Lin C-M (2008) A hybrid genetic algorithm for the re-entrant flow-shop scheduling problem. Expert Syst Appl 34:570–7
9. Chen J-S, Pan JC-H, Wu C-K (2007) Minimizing makespan in reentrant flow-shops using hybrid tabu search. Int J Adv Manuf Technol 34:353–61
10. Chiang T-C (2013) Enhancing rule-based scheduling in wafer fabrication facilities by evolutionary algorithms: review and opportunity. Comput Indust Eng 64:524–35
11. Choi H-S, Kim J-S, Lee D-H (2011) Real-time scheduling for reentrant hybrid flow shops: a decision tree based mechanism and its application to a TFT-LCD line. Expert Syst Appl 38:3514–21
12. Choi S-W, Kim Y-D (2008) Minimizing makespan on an m-machine re-entrant flowshop. Comput Oper Res 35:1684–96
13. Costa A, Cappadonna FA, Fichera S (2013) A dual encoding-based meta-heuristic algorithm for solving a constrained hybrid flow shop scheduling problem. Comput Indust Eng 64:937–58
14. Demirkol E, Uzsoy R (2000) Decomposition methods for the reentrant flow shops with sequence-dependent setup times. J Sched 3(3):155–77
15. Dudewicz EJ, Mishra SN (1988) Modern mathematical statistics, Wiley series in probability and mathematical statistics. Wiley, Singapore
16. Dugardin F, Yalaoui F, Amodeo L (2010) New multi-objective method to solve reentrant hybrid flow shop scheduling problem. Eur J Oper Res 203:22–31
17. Glover F, Laguna M (1997) Tabu search. Kluwer Academic Publishers, Dordrecht/London
18. Gupta AK, Sivakumar AI (2006) Job shop scheduling techniques in semiconductor manufacturing. Int J Adv Manuf Technol 27:1163–69
19. Gupta JND (1988) Two-stage, hybrid flowshop scheduling problem. J Oper Res Soc 39(4):359–64
20. Jing C, Huang W, Tang G (2011) Minimizing total completion time for re-entrant flow shop scheduling problems. Theor Comput Sci 412:6712–19
21. Joo BJ, Choi YC, Xirouchakis P (2013) Dispatching rule-based algorithms for a dynamic flexible flow shop scheduling problem with time-dependent process defect rate and quality feedback. Proced CIRP 7:163–8
22. Kang Y-H, Kim S-S, Shin HJ (2007) A scheduling algorithm for the reentrant shop: an application in semiconductor manufacture. Int J Adv Manuf Technol 35(5/6):566–74
23. Kis T, Pesch E (2005) A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. Eur J Oper Res 164:592–608
24. Laarhoven PJM, Aarts EHL (eds) (1987) Simulated annealing: theory and applications. Kluwer Academic Publishers, MA, USA
25. Lee G-C (2014) Real time order flowtime estimation methods for two-stage hybrid flowshops. Omega 42:1–8
26. Li J-Q, Pan Q-k, Wang F-T (2014) A hybrid variable neighborhood search for solving the hybrid flow shop scheduling problem. Appl Soft Comput 24:63–77
27. Lin D, Lee CKM (2011) A review of the research methodology for the re-entrant scheduling problem. Int J Prod Res 49(8):2221–42
28. Linn R, Zhang W (1999) Hybrid flow shop scheduling: a survey. Comput Indust Eng 37:57–61

29. Middendorf M, Timkovsky VG (2002) On scheduling cycle shops: classification, complexity and approximation. J Sched 5(2):135–69
30. Naderi B, Ruiz R, Zandieh M (2010) Algorithms for a realistic variant of flowshop scheduling. Comput Oper Res 37:236–46
31. Naderi B, Zandieh M, Roshanaei V (2009) Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum tardiness. Int J Adv Manuf Technol 41:1186–98
32. Pan Q-K, Wang L, Li J-Q, Duan J-H (2014) A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. Omega 45:42–56
33. Quadt D, Kuhn H (2007) A taxonomy of flexible flow line scheduling procedures. Eur J Oper Res 178:686–98
34. Ribas I, Leisten R, Framiñan JM (2010) Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Comput Oper Res 37:1439–54
35. Ruiz R, Şerifoğlu FS, Urlings T (2008) Modeling realistic hybrid flexible flowshop scheduling problems. Comput Oper Res 35:1151–75
36. Ruiz R, Maroto C (2006) A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. Eur J Oper Res 169:781–800
37. Ruiz R, Vázquez-Rodríguez JA (2010) The hybrid flow shop scheduling problem. Eur J Oper Res 205:1–18
38. Steiner G, Xue Z (2005) Scheduling in reentrant robotic cells: algorithms and complexity. J Sched 8:25–48
39. Wang H (2005) Flexible flow shop scheduling: optimum, heuristics and artificial intelligence solutions. Expert Syst 22(2):78–85
40. Wang S, Lin M (2013) A genetic algorithm for two-stage no-wait hybrid flow shop scheduling problem. Comput Oper Res 40:1064–75
41. Weng W, Wei X, Fujimura S (2012) Dynamic routing strategies for JIT production in hybrid flow shops. Comput Oper Res 39:3316–24
42. Zare HK, Fakhrzad MB (2011) Solving flexible flow-shop problem with a hybrid genetic algorithm and data mining: a fuzzy approach. Expert Syst Appl 38:7609–15