A job shop scheduling problem with human operators in handicraft production

Alessandro Agnetis\textsuperscript{a*}, Gianluca Murgia\textsuperscript{a} and Simone Sbrilli\textsuperscript{b}

\textsuperscript{a}Department of Information Engineering and Mathematical Sciences, University of Siena, Siena, Italy; \textsuperscript{b}Department of Enterprise Engineering, University of Rome “Tor Vergata”, Rome, Italy

\textit{(Received 25 January 2013; accepted 19 July 2013)}

This paper deals with complex job shop scheduling problems. A (typically large) number of elementary tasks has to be carried out, according to precedence constraints defined by a task graph. As typical of production environments such as handicraft production and task processing requires two different resources, i.e. machines and human operators. While for each operation a given machine is specified, there are in general more human operators capable of performing it. The problem is to assign the tasks to the operators, sequence them on each operator and sequence them on each machine so that the overall makespan is minimised. This scheduling problem is NP-hard even if the task graph consists of three chains (three-job job shop), and there are two fully skilled operators. We propose two heuristics for this scheduling problem, based on two different ways of decomposing the problem. An extensive computational experience allows a comparison between the heuristic solutions and the one obtained solving a mixed-integer programming formulation of the problem. The experiments show that close-to-optimal solutions can be obtained in reasonable time on a PC. Our model is applied to a case study from leather manufacturing, and we also show its use as a decision support tool in skill planning.

\textbf{Keywords:} handicraft production; heuristics; human operators; job shop

1. Introduction

Under growing market pressure, handicraft firms are improving their planning procedures in order to provide adequate customer service level and short lead times. This phenomenon particularly affects luxury goods, for which the customers’ expectations are higher, and, even more, ‘fashion-sensitive’ goods, which are typically subject to seasonal fashion and hardly predictable demand \cite{Caniato}. In fact, the risk of obsolescence, which affects luxury firms more than non-luxury ones because of the higher unit cost, may push them to adopt a make-to-order approach \cite{Caniato2}. The correct implementation of this approach requires a strict coordination among the different production phases. This is especially true when such phases are partially or totally outsourced to a wide network of very small and specialised firms, as occurs in the Italian luxury fashion context \cite{Djelic}. In such contexts, the outsourcer and the outsourcees have to define an integrated production plan that establishes also the due dates for all the production phases of each batch.

In this paper, we develop an efficient algorithm that supports the daily production planning of the outsourcee. Our algorithm solves a complex scheduling problem characterized by the presence of two different resources, machines and operators, that have to be synchronised in order to minimise the makespan of the batch. We have tested our approach on a set of random instances and in a real case study from a typical Italian leather handicraft firm.

The plan of the paper is as follows. The problem and related literature are described in detail in Section 2. In Section 3, a mathematical model and the related notation are introduced. Two heuristic approaches are described in Sections 3.1 and 3.2, respectively. Experimental results on random instances are presented in Section 4, while the case study is described in Section 5. Finally, in Section 6 we present the conclusions and the future developments of our study.

2. Problem description and literature review

Many handicraft firms deal with complex scheduling problems in the management of their assembly line. In fact, the tasks needed to assemble a single product may be many and subject to several precedence constraints. The process is often characterised by the presence of different types of machines, i.e. general purpose machines (e.g. workbenches) and dedicated purpose machines (e.g. numerical control sewing machines). Nevertheless, the most important peculiarity of the assembly process in the handicraft firms is the fact that the tasks cannot be performed without the presence of an operator, having

\*Corresponding author. Email: agnetis@dii.unisi.it

© 2013 Taylor & Francis
an adequate skill. The skill set of an operator in handicraft firms is generally related to his/her seniority and specialisation. For instance, an apprentice may be able to perform only a limited subset of assembly tasks. On the other hand, a more experienced operator may be able to perform many tasks, possibly requiring different machines. In general, the assembly process can be correctly performed only if each task is processed by the right machine and the right operator. In a scheduling perspective, this means that each task requires at the same time two different resources, i.e. an operator and a machine. In order to minimise the batch makespan, it is therefore necessary to correctly assign the tasks to the operators, and schedule the tasks on the machines in the job shop.

To the best of our knowledge, only few papers in the literature address job shop problems considering two resource types. In the past years, many papers dealt with some scheduling problems with operators, both in open shops and in parallel-machine contexts. Baki and Vickson (2004) consider the problem of one-resource, two-machine open shop, showing that the problem is NP-hard even for a single operator. Kellerer and Strusevich (2003), Glass, Shafransky, and Strusevich (2000) and Hall, Potts, and Sriskandarajah (2000) discuss parallel-machine scheduling problems in which, besides a dedicated machine, a common resource (operator) is required to process each job. Agnetis et al. (2010) address the job shop problem with \( n \) jobs, with the constraint that each task requires one operator, and there are \( p < n \) identical operators, each enabled to process any task on any machine. This scheduling problem is NP-hard even if the precedence constraints among tasks consist of three chains (three-job job shop). In recent years, Mencía et al. (2011) proposed artificial intelligence algorithms to address the problem. Sierrra, Mencía, and Varela (2011) proposed instead a scheduling generation scheme for the problem and a dedicated heuristic where the objective is total flow time minimisation.

In the literature, operator assignment problems are widely investigated in the context of employee timetabling problems. Some of these papers address problems related to the one investigated in this paper. Recently, Guyon et al. (2010) integrate timetabling and production scheduling problems. Their problems is to assign interruptible jobs with time windows to appropriate operators. No precedence relationships are considered and their objective function is the minimisation of labour cost. In Guyon et al. (2012), the timetabling problem is integrated in a job shop environment. In this paper, the authors consider a scheduling problem in which every machine is assigned to an operator that works on a three-shift system, and the objective is to assign a certain operator to a certain shift so that costs are minimised. The makespan is bounded by a maximum value and each operator is able to use a subset of machines. Artigues et al. (2009) also consider the operators’ allocation to different shifts in a job shop production line, using constraint programming to compute the solution.

While those papers address operator-to-machine or operator-to-shift assignment decisions, one key issue of our model is assigning operators to tasks. In fact, in our assembly systems, operators may have different skills, so that, even if various tasks require the same machine, an operator may be able to perform only a subset of these tasks.

The generalised problem of multiple resource scheduling has been also investigated in the literature due to the practical importance (Driebel and Mönch 2012). Ruiz-Torres, Ablanedo-Rosas, and Otero (2012) propose a scheduling approach in the pharmaceutical industry that deals with the assigning tasks to technicians in the quality control phase and scheduling them in order to minimise the total flow time. Wu et al. (2011) propose an interesting genetic algorithm for solving a simultaneous multiple resources scheduling problem in which every machine can be seen as set of multiple resources that must be aligned in order to process a job.

3. Our approach: model and heuristic methods

We address the problem of minimising makespan in a job shop, assigning the operators to tasks and sequencing the tasks on the machines. Each task has to be processed by a given machine. This is quite common in handicraft firms, in which each task has to be performed by the only special-purpose machine that can process it. Indeed, some tasks might be performed by general-purpose machines, but we assume that even in these cases the machine assignment is given because of layout and balancing constraints. Hence, in order to optimise the production process, it is necessary to correctly assign the operators to the tasks. This is especially true when the number of operators is much lower than the number of machines, as it is typical of many handicraft firms.

The problem can be formally described as follows. An assembly system is given, consisting of a set \( M \) of \( q \) machines and a set \( O \) of \( p \) operators. A set \( T \) of \( n \) different tasks has to be carried out. Task \( i \) \( (i = 1, \ldots, n) \) requires a processing time \( p_{ti} \), as well as two resources: (i) a given machine \( m_i \in M \) and (ii) a human operator \( o \in O \), chosen among those who are able to perform it. Both resources are released upon task completion. The skills of the different operators are specified by a skill matrix \( A \), whose element \( a_{io}, i \in T, o \in O \) is defined as:

\[
a_{io} = \begin{cases} 
1 & \text{if operator } o \text{ has the skill to process task } i \\
0 & \text{otherwise}
\end{cases}
\]
We denote by $T_o$ the task set that can be performed by operator $o$, and $O_i$ the set of operators that can perform task $i$ (both assumed non-empty). We assume that the processing time of a single task is independent from the operator in charge of its execution. Tasks cannot be preempted. A directed task graph $G(T, A)$ specifies precedence relations among tasks. Nodes correspond to tasks, and arc $(i, j) \in A$ means that task $i$ must be completed before task $j$ starts. In many applications, this graph has a tree structure, the root corresponding to the final assembly task. The objective is to find a task-to-operator allocation along with an overall task schedule so that the makespan is minimised. We refer to this problem as Job Shop with Skilled Operators (JSSO). The output of JSSO consists therefore in two sets of sequences, one for each machine in the system and one for each operator.

JSSO is a generalisation of the classical job shop scheduling problem (JSSP). Indeed, in the context of handicraft leather manufacturing, the assembly of each of the main components of a product (e.g. a batch of luxury bag) can be seen as a single job, having a particular routing. The components have to be assembled, and this leads to precedence constraints across different jobs. Moreover, a single component can visit the same machine many times during its routing, i.e. there is recirculation. Set-up times are not present or can be considered negligible with respect to processing times. Hence, using the classic Graham notation (Graham and Lawler 1979), JSSO generalizes $J|\text{prec, recr}|\text{C}_{\max}$.

We next give a mathematical formulation for JSSO. Let $S_i$ and $C_i$ be the start time and the finish time for task $i$, $i \in T$. The binary decision variable $x_{io}$ is equal to 1 if $i$ is assigned to operator $o \in O_i$. Another decision variable $y_{it}$ is equal to 1 if task $t$ starts after the completion of task $i$.

$$\min \ C_{\max} \ (1)$$

s.t

$$C_{\max} \geq C_i \ \forall i \in T \ (2)$$

$$S_i + p_i = C_i \ \forall i \in T \ (3)$$

$$C_i \leq S_i + M(1 - y_{it}) \ \forall i, t \in T, m_i = m_t \ (4)$$

$$C_i \leq S_i + M y_{it} \ \forall i, t \in T, m_i = m_t \ (5)$$

$$C_i \leq S_i + M(1 - y_{it}) + M(2 - x_{io} - x_{to}) \ \forall i, t \in T, \forall o \in O \ (6)$$

$$C_i \leq S_i + M y_{it} + M(2 - x_{io} - x_{to}) \ \forall i, t \in T, \forall o \in O \ (7)$$

$$S_i \geq C_i \ \forall (i, t) \in A \ (8)$$

$$\sum_{i \notin T_o} x_{io} = 0 \ \forall o \in O \ (9)$$

$$\sum_{o \in O_i} x_{io} = 1 \ \forall i \in T \ (10)$$

$$S_i \geq 0 \ \forall i \in T \ (11)$$

$$x_{io} \in \{0, 1\} \ \forall i \in T, \forall o \in O \ (12)$$

Objective function (1) and constraints (2) ensure that we minimise the completion time of the last scheduled task (makespan). Constraints (3) enforce non-preemption. Constraints (4)–(5) impose that a machine cannot process more than one task at a time, i.e. the logic proposition

$$(C_i \leq S_i) \lor (C_i \leq S_t)$$

must be true for any two tasks $i, t \in T$ such that $m_i = m_t$. Constant $M$ is large enough to ensure that the constraints hold for any reasonable value of the considered variables. Constraints (6)–(7) impose the logic condition

$$(C_i \leq S_t) \lor (C_i \leq S_i) \iff x_{io} = x_{to} = 1$$

If two different tasks $i, t$ are assigned to the same operator $o \in O$, they cannot overlap. The precedence relationships in $G$ are modelled by constraints (8). Constraints (9) ensure that an operator $o$ cannot perform task $i$ if he/she does not have an adequate skill, i.e. if $i \notin T_o$. Finally, constraints (10) impose that each task $i$ must be assigned to an operator of $O_i$. Note that, unlike in classical JSSP, the solution must specify not only the start and finish time of each task, but also the operator in charge of its execution. In order to better explain the structure of the proposed problem, a simple example is presented.
Example 3.1 Let us consider a small JSSO example with \( n = 7 \) tasks, \( p = 2 \) operators, and \( q = 3 \) machines. The processing time and required machine for each task are reported in Table 2. The precedence graph is depicted in Figure 1, while the skill matrix \( \mathbf{A} \) is shown in Table 1.

The optimal solution of this small example is shown in Figure 2. The different colors on the Gantt charts denote the task-to-operator assignment. In order to better explain the role of the skill matrix, Figure 2(b) shows the solution in case of fully skilled operators. Note that, in this case, the operators are not a critical resource.

Figure 2(a) shows that, due to the skill matrix \( \mathbf{A} \), Task 6 starts after the completion of Task 3. In fact, only operator A is able to process them. This, of course, increases the final makespan.
Solving formulation (1)–(12) to optimality can be very time and memory consuming. The idea of using a heuristic approach appears therefore reasonable. Decomposition methods can be a powerful tool to address such double-resource type of problems (Ovacik and Uzsoy 1997). In what follows we focus on designing possible decomposition algorithms for the problem.

In JSSO, the three distinct key decisions are:

1. Assign tasks to operators
2. Sequence tasks on each operator
3. Sequence tasks on each machine

JSSO consists on specifying all these decisions concurrently. Decomposition methods address them separately, giving rise to various possible decomposition algorithms.

A very powerful tool in decomposition methods for job shops (without operators) is the well-known Shifting Bottleneck (SB) heuristic (Adams, Balas, and Zawack 1988). We briefly recall here its basic operation. In what follows, we let $T(m)$ denote the set of tasks requiring machine $m$. SB uses a current task graph $G$ which is initialized as $G$. With no loss of generality, we add two dummy nodes $O$ and $F$ on $G$, representing respectively the start and finish of the whole set of tasks.

Let $K$ be the length of the critical path from $O$ to $F$ on $G$. SB considers each of the $q$ machines in turn. For each machine $m$, an instance of the single-machine problem $1|\text{pmtn}|L_{\text{max}}$ is defined in which only the tasks of $T(m)$ appear. The release and due date of each task $i$ are the length of the longest path from $O$ to $i$ on $G$ and, respectively, $K$ minus the length of the longest path from $i$ to $F$ on $G$. The machine for which the optimal value of $L_{\text{max}}$ is the highest, say, $m^\ast$, is considered the bottleneck, and the sequencing prescribed by the optimal solution of the corresponding $1|\text{pmtn}|L_{\text{max}}$ instance is adopted. This means that the arcs corresponding to such sequencing are added to $G$, and as a consequence the value of the critical path length is updated. Then, the sequencing on machine $m^\ast$ is frozen, and the whole procedure is repeated considering the other $q-1$ machines. This goes on until all machines are sequenced. At the end, the length of the critical path on $G$ is the makespan of the whole task set.

Indeed, SB refinements include the possibility of resequencing (i.e. at each step unfreeze and modify the arcs of some already scheduled machine, if this can improve the current value of $K$). However, incorporating resequencing issues in the modified SB versions employed by our heuristics would considerably increase their computational burden. Hence, this topic is left for further investigation.

In our computational experiments, each instance of $1|\text{pmtn}|L_{\text{max}}$ is solved by a combinatorial branch and bound algorithm, in which the preemptive relaxation $1|\text{pmtn}|L_{\text{max}}$ is used to compute a lower bound at each node.

### 3.1 Heuristic STA-OMSB

Our problem is different from classical job shop in that each task must be assigned to an operator. Hence, one idea is to extend the SB approach by first assigning an operator to each task, and then treating machines and operators homogeneously. With respect to the three above key decisions, this approach consists on addressing point 1 first, and then points 2 and 3 concurrently. More precisely, we get the following heuristic (referred to as STA-OMSB):

(i) Given the task set, assign the tasks to the operators so that their workload is balanced (STA, Static Task Assignment);

(ii) Sequence the tasks on the operators and the machines by using the shifting bottleneck algorithm (OMSB, Operator + Machine Shifting Bottleneck).

In view of the heuristic context, it appears reasonable to perform STA by means of a very efficient heuristic for multiprocessor scheduling, i.e. lowest fit decreasing (Graham and Lawler 1979). Such rule (which guarantees an approximation of 4/3) simply consists on ordering the tasks from the longest to the shortest, and successively assigning them to the operators (that can perform the task), always assigning the next task to the currently least loaded operator.

For what concerns OMSB, it consists on running the classical SB algorithm, provided that now operators and machines are considered individually. We observe that the fact that a task requires both an operator and a machine can be easily handled by SB. In fact, consider the following situation. There are four tasks, $a, b, c, d$, for which operators and machines are as indicated in Table 3. Suppose that SB has first solved the sequencing problem for operator A, setting $a \prec b$ and, consequently, adding the disjunctive arc from $a$ to $b$. Then SB sequenced machine 2, setting $b \prec c$, and then operator B, setting $c \prec d$. Now it comes to sequencing machine 1. Actually, SB might set $d \prec c$, thus creating a cycle in the disjunctive graph. However, if this happens, then also the sequence in which $a \prec d$ is optimal for the instance of $1|\text{pmtn}|L_{\text{max}}$ associated with machine 1, since $d$ has both release date and due date larger than those of $a$. Hence, even if the addition of a precedence constraint creates a cycle in the disjunctive graph, such cycle can be removed simply by reversing the arc that generated it.

Heuristic STA-OMSB is summarised below.
Table 3. Example for OMSB.

| Task | Operator | Machine |
|------|----------|---------|
| a    | A        | 1       |
| b    | A        | 2       |
| c    | B        | 2       |
| d    | B        | 1       |

Algorithm 1. STA-OMSB heuristic pseudocode

Require: The task graph $G$, an operator set $O$ and a machine set $M$;

1: initialisation;
2: $G = G$;
3: Solve the assignment problem: balance operators’ workload (STA);
4: $R = (O \cup M)$;
5: $R_0 = \emptyset$;
6: while $R_0 \neq R$ do
7: Calculate release dates and due dates on $G$ with Ford–Bellman algorithm;
8: for $v \in (R \setminus R_0)$ do
9: Solve $1|r_j, recr|L_{\max}$ for the resource $v$;
10: Get the corresponding $L_{\max}$ and schedule $\sigma^v$;
11: end for
12: $h = \arg \max_v L_{\max}(v)$;
13: $R_0 = R_0 \cup \{h\}$;
14: Given $\sigma^h$, insert corresponding arcs into $G$;
15: end while
return Schedule $\sigma = \{\sigma^v, v \in R\}$;

3.2 Heuristic MSB-DOS

Heuristic STA-OMSB allows to exploit the well-known effectiveness of the shifting bottleneck paradigm, treating operators and machines homogeneously. However, it requires to decide a task-to-operator assignment beforehand. If we decide it only on the basis of operator workload balancing, the resulting assignment may be unsatisfactory. An alternative idea consists on first sequencing the machines, and then deciding the task-to-operator assignment, in a way that accounts for the actual structure of the precedence graph. With reference to the key decisions in Section 3, we now first address decision 3, and then 1 and 2 concurrently. We get the following heuristic (referred as MSB-DOS):

(i) Given the task set, sequence the tasks on each machine by using the shifting bottleneck procedure (MSB, Machine Shifting Bottleneck);

(ii) Assign the tasks to the operators and schedule them by a suitable algorithm (DOS, Dynamic Operators Sequencing).

Step (i) is carried out by the standard shifting bottleneck heuristic applied to machines only. Thus, the output of step (i) specifies a task sequence for each machine, and the corresponding arcs are added to the disjunctive graph $G$. Operators have not been taken into account yet.

Step (ii) is carried out by means of an algorithm that iteratively assigns groups of tasks to the operators and schedules them. The idea is the following. At each iteration, there is a set $U$ of tasks which are not yet assigned to any operator, and having no unscheduled predecessor on the current graph $G$. At the beginning, $U$ is initialised with the tasks having no predecessors on the graph $G$ returned by step (i).

DOS heuristic works as follows. While there is at least one task to be scheduled, the current set $U$ of unscheduled tasks without unscheduled predecessors is computed. Then, we assign these tasks to the operators. Unlike what done in STA-OMSB, we perform the assignment taking into account the topological information of the current disjunctive graph. To this aim, we associate with each task $t$ a positional weight $w_t$, equal to the length of the longest path in $G$ starting from node $t$. The idea (which is widely used in assembly line balancing problems, Baybars (1986)) is that a large positional weight reflects the topological importance of a task, so that it may be profitable to assign tasks having large positional weights to different
operators. Hence, we perform lowest-fit-decreasing on the tasks in $U$, using positional weights instead of task lengths. This yields an assignment of tasks to operators for all tasks in $U$, let $U(o)$ be the set of tasks in $U$ assigned to operator $o$. At this point, separately for each set $U(o)$, we solve an instance of $1|r_j|L_{\text{max}}$, in which, for each task $t$, release and due dates are defined as in a shifting bottleneck context (as recalled in Section 3). In this way, for each operator $o$ the sequencing $\sigma_{U(o)}$ of the tasks in $U(o)$ is computed and appended to the current schedule $\sigma_o$ for operator $o$. Thereafter, the new set $U$ is computed and so on, until all tasks are assigned and sequenced.

The heuristic MSB-DOS is summarized in Algorithm 2, in which $D$ is the set of assigned and scheduled tasks, and $\circ$ indicates concatenation.

The two proposed heuristic approaches are summarised in Figure 3.

**Algorithm 2.** MSB-DOS heuristic pseudocode

```plaintext
Require: The task graph $G = (T, A)$, an operator set $O$ and a machine set $M$
1: initialisation:
2: $G = G$
3: Schedule the tasks on machines using shifting bottleneck and get the final disjunctive graph $G$;
4: $D = \emptyset$
5: $\sigma_o = \emptyset \ \forall o \in O$
6: while $D \neq T$ do
7: Update task release dates and due dates on $G$ with Ford–Bellman algorithm;
8: Compute set $U$;
9: Assign the tasks in $U$ to the available operators;
10: for $o \in O$ do
11: Solve the corresponding $1|r_j|L_{\text{max}}$ problem ;
12: Get the corresponding schedule $\sigma_{U(o)}$;
13: Update the schedule: $\sigma_o = \sigma_o \circ \sigma_{U(o)}$
14: end for
15: $D = D \cup U$
16: end while
return schedule $\sigma = [\sigma_o, o \in O]$;
```

4. Computational experiments

In order to evaluate the effectiveness of the proposed heuristics, we designed a set of experimental scenarios, each defined by a given number of tasks ($n$), operators ($p$) and machines ($q$).

For each triple $(n, p, q)$, we generated a set of 10 instances. For each instance, a task graph was generated having a similar topology to real-life assembly trees, as the one described in Section 5. More precisely, the $n$ tasks are distributed
Table 4. Results of the experiments.

| Characteristics | MSB-DOS | STA-OMSB | Cplex | LB GAP |
|-----------------|---------|----------|-------|--------|
| Tasks | Operators | Machines | p/q | Gap (%) | Time | opt. | Gap (%) | Time | opt. | Time | inf. | MSB-DOS (%) | STA-OMSB (%) |
| 100 | 10 | 15 | 0.67 | 0.9 | 13.6 | 4 | 8.5 | 43.0 | 2 | 40.7 | 0 | – | – |
| 100 | 10 | 20 | 0.50 | 0.7 | 3.4 | 6 | 5.3 | 5.5 | 0 | 42.1 | 0 | – | – |
| 150 | 10 | 15 | 0.67 | 1.2 | 8.3 | 2 | 5.7 | 14.4 | 0 | 223.9 | 0 | – | – |
| 150 | 10 | 20 | 0.50 | 0.4 | 8.2 | 7 | 5.4 | 13.1 | 0 | 138.6 | 0 | – | – |
| 150 | 15 | 30 | 0.50 | 0.2 | 8.3 | 6 | 2.4 | 8.7 | 1 | 190.5 | 0 | – | – |
| 150 | 15 | 50 | 0.30 | 0.0 | 9.6 | 9 | 1.9 | 11.2 | 2 | 129.3 | 0 | – | – |
| 150 | 25 | 50 | 0.50 | 0.3 | 17.2 | 5 | 3.1 | 46.1 | 1 | 130.2 | 0 | – | – |
| 200 | 15 | 30 | 0.50 | 0.4 | 42.1 | 7 | 1.1 | 29.9 | 1 | 1727.1 | 8 | 0.2 | 1.9 |
| 200 | 15 | 50 | 0.30 | 1.3 | 42.9 | 5 | 3.1 | 82.7 | 0 | 475.8 | 8 | 0.3 | 3.6 |
| 200 | 20 | 30 | 0.67 | 0.4 | 23.4 | 6 | 1.6 | 19.2 | 1 | 913.3 | 0 | – | – |

Among a random number of branches, chosen between 3 and 6, so that the number of tasks is balanced among the branches. Finally, each branch is connected to the next branch. In particular, the last task of each branch is connected to a randomly chosen task of the next branch, so that the task graph remains acyclic.

In real-life applications, tasks can often be partitioned into operating skill classes (Section 5) so that all the tasks of the same class require a certain skill. In such context, the most critical features appear to be the total number of tasks n and the number of operators p, less so the number of branches. In fact, preliminary experiments showed that similar results (in terms of computational efficiency) were obtained for instances with different numbers of branches but having the same n and the same skill matrix A. Actually, in terms of solution quality, the number of operators in the various skill classes is extremely relevant, as confirmed by the experiments that investigate the criticality of skill classes, illustrated in detail in Section 4.

Task processing times are uniformly distributed in the range [1, 100]. For each instance, a skill matrix has been randomly generated, by ensuring that every task has at least one operator able to process it. Finally, for each task we randomly associated the machine in such a way that machine workloads were roughly balanced.

STA-OMSB and MSB-DOS were run on each instance. We have also run Cplex 12.2 on the formulation (1)–(12) proposed in Section 3 with the time limit of 3600 s. In what follows, we report the results of the experiments, comparing the performance of the heuristic approaches and Cplex. In some instances, Cplex was not able to find a feasible solution within the time limit. For these cases, the quality of the heuristic solution was assessed using the best available lower bound provided by Cplex. All tests were performed on a AMD Athlon II 2.70 GHz using Matlab 2009b.

For each scenario, Table 4 shows the average values (over 10 instances) of the following quantities.

- MSB-DOS gap: the gap (in %) between the value of the schedule produced by MSB-DOS and the optimal value (when an optimal solution is found);
- MSB-DOS time: CPU time spent by MSB-DOS;
- opt MSB-DOS: number of instances (out of 10) for which MSB-DOS found a certified optimal solution;
- STA-OMSB gap: the gap (in %) between the value of the schedule produced by STA-OMSB and the optimal value (when an optimal solution is found);
- STA-OMSB time: CPU time spent by STA-OMSB;
- opt STA-OMSB: number of instances (out of 10) for which STA-OMSB found a certified optimal solution;
- Cplex time: CPU time spent by Cplex when the optimal solution is found;
- inf Cplex: number of instances (out of 10) in which Cplex did not find a feasible solution within the time limit;
- MSB-DOS LB gap: the gap between the value of the schedule produced by MSB-DOS and the lower bound when the time limit was exceeded;
- STA-OMSB LB gap: the gap between the value of the schedule produced by STA-OMSB and the lower bound when the time limit was exceeded.

A few comments are in order.

- MSB-DOS outperforms STA-OMSB from both viewpoints of solution quality and CPU time. The average optimality gap over all instances is 0.6% for MSB-DOS vs. 3.7% for STA-OMSB. Actually, even for large instances (n = 200) the MSB-DOS gap was still 0.6% on the average. This suggests that a dynamic task-to-operator assignment is highly beneficial for solution quality. Also, the larger time spent by DOS with
respect to STA is balanced by the fact that MSB solves only $O(q)$ instances of $1|r_j|L_{\text{max}}$ vs. $O(q+p)$ instances solved by OMSB. We also notice that MSB-DOS actually finds the optimal solution in 60 instances out of 100, vs. 8 of STA-OMSB.

- Heuristic MSB-DOS (as well as STA-OMSB) requires significantly less CPU time than Cplex, in all experiments. Moreover, in many large instances Cplex fails finding a feasible solution.
- As problem size grows, the ratio between the CPU times required by Cplex and MSB-DOS increases. Actually, MSB-DOS is comparatively more efficient in large instances. Also, one can observe that, in the largest instances ($n = 200$), the higher the ratio $p/q$ (i.e. the more critical the human resources), the harder the instance for Cplex. So, the scenario $(200, 20, 30)$ appears indeed easier than $(200, 15, 50)$. On the contrary, the CPU time required by MSB-DOS is not significantly affected by this issue. We notice that in the scenarios with $n = 200$, Cplex was not able to find a feasible solution within the time limit for most of the instances with $p/q \leq 0.5$ (while MSB-DOS solved to optimality 12 out of 20 such instances).

These results suggest that MSB-DOS is a suitable tool for solving JSSO in realistic instances in handicraft manufacturing environments.

5. Case study

The heuristics presented in this paper were applied to data from Essetre Fashion Group s.r.l., an Italian small-size leather firm, outsourcee of major international luxury brands. This firm is located in Tuscany, in the Amiata area, where leather manufacturing started in the 1950s and now involves more than 100 firms, almost all of small size. Essetre Fashion Group s.r.l. has been established in 2008 and its facilities are capable of performing all necessary phases for the production of a leather bag, including pattern making, cutting, preparation, dyeing, assembly and packaging. The case study refers to the main phase of the production process, i.e. the assembly phase, in which the semi-finished products are combined together. The assembly of a single bag requires 97 different operations. Since bags are produced in batches of 10 units each, each task consists on performing the same operation consecutively on the 10 bags. Each task is characterized by a processing time that varies from 10 to 150 min. Various semi-finished parts of a bag correspond to branches in the task graph, which also contains several precedence relationships across different branches.

In the system, there are $p = 13$ operators and $q = 23$ workstations, namely 7 workbenches and 16 dedicated machines (all modelled as machines). Even if in principle some tasks may be performed by various workbenches, physical constraints suggest to pre-assign the tasks, so that the material flow is smoother. Figure 4 shows the task graph for the case study, in which each task is indicated by its ID number. Tasks processed in a workbench are depicted as circles, while those processed in a dedicated machine are depicted as squares.²

![Figure 4. Task graph for the case study.](image-url)
The assembly tasks can be classified into 14 different Operating Skill Classes (OSC), depending on the skill required. The skill matrix of all the operators was obtained through a dedicated assessment session with the production manager. In Table 5, entry \((o, c)\) equals 1 if the operator \(o\) has the skill to perform tasks in class \(c\). We assume that the processing time of a task is the same for all skilled operators. Note that in this problem some classes of tasks only have one operator skilled for it. In general, this may be a crucial aspect because these operators may become the bottleneck of the system.

Both heuristics and Cplex were run on the instance. All approaches required few seconds of CPU. The heuristic MSB-DOS provides a solution having makespan 1042 min, i.e. 3.2% higher than the optimum value 1009.

From a management perspective, one of the most important issues is the training of the operators. Having a set of fully skilled operators (i.e. operators for which \(T_o = \bar{T}\)) grants the shortest completion time, but it requires a high investment in operators training. Having a quick tool (such as MSB-DOS) for makespan estimation allows one to perform alternative experiments, aimed at detecting which skills are more critical. This analysis can support the managers in optimising the training programme of the operators.

We next illustrate this concept using the case study example. In order to detect the most critical OSC, one can consider the following scenarios. In scenario \(S_0\), all operators are fully skilled. In scenario \(S_c\), \(p - 1\) operators are able to perform all but one OSC, called \(c\), while the \(p\)-th operator is fully skilled, so that he/she is the only one enabled to perform tasks of OSC \(c\), \(c = 1, \ldots, 14\).

### Table 6. Measure of OSC criticality.

| OSC | \(\left(\frac{C_{\text{max}}^c}{C_{\text{max}}^0}\right) / C_{\text{max}}^0\) (%) |
|-----|--------------------------------------------------|
| 1   | 6.4                                             |
| 2   | 1.8                                             |
| 3   | 0.0                                             |
| 4   | 11.0                                            |
| 5   | 0.0                                             |
| 6   | 0.0                                             |
| 7   | 0.0                                             |
| 8   | 1.8                                             |
| 9   | 0.0                                             |
| 10  | 0.0                                             |
| 11  | 0.9                                             |
| 12  | 1.1                                             |
| 13  | 2.3                                             |
| 14  | 3.7                                             |
For all scenarios, we compute the makespan using MSB-DOS, let $C_{c_{\text{max}}}^c$ be the value obtained in scenario $S_c$, $c = 0, 1, \ldots, 14$. The ratio $(C_{c_{\text{max}}}^c - C_0^c)/C_0^c$ can be taken as a measure of the criticality of the OSC $c$. Table 6 depicts the value of this ratio for each $c$. The most critical skill classes turns out to be OSC 4 and OSC 1, for which the relative makespan increases are 11 and 6.4%, respectively. The values of these ratios can be compared vs. the cost of training additional people for the respective skill class.

6. Conclusions

Human operators represent the most critical resource in handicraft firms, since they affect product quality, but also the service level provided to customers. Nevertheless, literature on job shop scheduling with human operators is not yet fully developed. One of the main lacks is related to the issue of operator skills. These are generally considered identical for all the operators, even if, in real production systems, they may differ radically between an experienced operator and an apprentice. This fact particularly affects handicraft firms, but also many industrial ones, where certain tasks can be performed only by a skilled operator.

In this paper, we propose two heuristics that solve job shop scheduling problems, taking into account also the skill set of each operator. In particular, heuristic MSB-DOS obtains close-to-optimal solutions in reasonable time on a PC. So, our heuristics can be used to compute an efficient task schedule. Besides, these heuristics can be used to support the development of the training programme of the operators, because they allow the detection of the most critical skills and their impact on the makespan.

Future research on JSSO is likely to address various topics, including:

- Extension of our approach to the more general case of cyclic production, in which different product batches are in process at the same time. In this context, we expect the topology of the assembly tree to significantly affect the complexity of the problem. Moreover, in this context the problem of sizing human resources is even more relevant and hence our approach expectedly more useful.
- Possible refinements of the proposed heuristics, including the introduction of resequencing issues.
- The application of metaheuristic approaches. These have not yet been applied to JSSO, but might turn out to be effective, in view of the complexity of the problem.

Acknowledgements

The authors gratefully acknowledge Essetre Fashion Group s.r.l. for its availability and openness, crucial for the development of the case study. Riccardo Certosini has provided valuable contributions in an earlier study of leather production planning.

Notes

1. Details on the generated instances are provided in the supplementary material.
2. Details on each task, including a brief description and the processing time, are provided in the supplementary material.

References

Adams, J., E. Balas, and D. Zawack. 1988. “The Shifting Bottleneck Procedure for Job Shop Scheduling.” Management Science 34 (3): 391–401.
Agnetis, A., M. Flamini, G. Nicosia, and A. Pacifici. 2010. “A Job-shop Problem with One Additional Resource Type.” Journal of Scheduling 14 (3): 225–237.
Artigues, C., M. Gendreau, L. M. Rousseau, and A. Vergnaud. 2009. “Solving an Integrated Employee Timetabling and Job-shop Scheduling Problem via Hybrid Branch-and-bound.” Computers & Operations Research 36 (8): 2330–2340.
Baki, M. F., and R. G. Vickson. 2004. “One-operator, Two-machine Open Shop and Flow Shop Problems with Setup Times for Machines and Weighted Number of Tardy Jobs Objective.” Optimization Methods and Software 19 (2): 1–24.
Baybars, I. 1986. “A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem.” Management Science 32 (8): 909–932.
Caniato, F., M. Caridi, C. M. Castelli, and R. Golini. 2009. “A Contingency Approach for SC Strategy in the Italian Luxury Industry: Do Consolidated Models Fit?” International Journal of Production Economics 120 (1): 176–189.
Caniato, F., M. Caridi, C. M. Castelli, and R. Golini. 2011. “Supply Chain Management in the Luxury Industry: A First Classification of Companies and their Strategies.” International Journal of Production Economics 133 (2): 622–633.
Djelic, M. L., and A. Aimamo. 1999. “The Coevolution of New Organizational Forms in the Fashion Industry: A Historical and Comparative Study of France, Italy, and the United States.” Organization Science 10 (5): 622–637.
Driebel, R., and L. Mönch. 2012. “An Integrated Scheduling and Material-handling Approach for Complex Job Shops: A Computational Study.” *International Journal of Production Research* 50 (20): 37–41.

Glass, C. A., Y. M. Shafransky, and V. A. Strusevich. 2000. “Scheduling for Parallel Dedicated Machines with a Single Server.” *Naval Research Logistics* 47 (4): 304–328.

Graham, R. L., and E. L. Lawler. 1979. “Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey.” *Annals of Discrete Mathematics* 5: 287–326.

Guyon, O., P. Lemaire, É. Pinson, and D. Rivreau. 2010. “Cut Generation for an Integrated Employee Timetabling and Production Scheduling Problem.” *European Journal of Operational Research* 201 (2): 557–567.

Guyon, O., P. Lemaire, É. Pinson, and D. Rivreau. 2012. “Solving an Integrated Job-shop Problem with Human Resource Constraints.” *Annals of Operations Research* May: 1–25.

Hall, N. G., C. N. Potts, and C. Sriskandarajah. 2000. “Parallel Machine Scheduling with a Common Server.” *Discrete Applied Mathematics* 102 (3): 223–243.

Kellerer, H., and V. A. Strusevich. 2003. “Scheduling Parallel Dedicated Machines Under a Single Non-shared Resource.” *European Journal of Operational Research* 147 (2): 345–364.

Mencía, R., M. Sierra, C. Mencía, and R. Varela. 2011. “Genetic Algorithm for Job-shop Scheduling with Operators.” In *New Challenges on Bioinspired Applications, Volume 6687 of Lecture Notes in Computer Science*, edited by J. Ferrández, J. A. Sánchez, F. de la Paz and F. Toledo, 305–314. Berlin: Springer.

Ovacik, I. M., and R. Uzsoy. 1997. *Decomposition Methods for Complex Factory Scheduling Problems*. Dordrecht: Kluwer Academic.

Ruiz-Torres, A. J., J. H. Ablanedo-Rosas, and L. D. Otero. 2012. “Scheduling with Multiple Tasks per Job – The Case of Quality Control Laboratories in the Pharmaceutical Industry.” *International Journal of Production Research* 50 (3): 37–41.

Sierra, M., C. Mencía, and R. Varela. 2011. “Optimally Scheduling a Job-shop with Operators and Total Flow Time Minimization.” In *Proceedings of the 14th International Conference on Advances in Artificial Intelligence: Spanish Association for Artificial Intelligence*, 193–202, CAEPIA 11, Berlin: Springer-Verlag.

Wu, J. Z., X. C. Hao, C. F. Chien, and M. Gen. 2011. “A Novel Bi-vector Encoding Genetic Algorithm for the Simultaneous Multiple Resources Scheduling Problem.” *Journal of Intelligent Manufacturing* 23 (6): 2255–2270.