RAIRO-Oper. Res. 55 (2021) S727–S740
https://doi.org/10.1051/ro/2020006

**RAIRO Operations Research**
www.rairo-ro.org

# AN FPTAS FOR JUST-IN-TIME SCHEDULING OF A FLOW SHOP MANUFACTURING PROCESS WITH DIFFERENT SERVICE QUALITY LEVELS

Amir Elalouf*

**Abstract.** This paper addresses the problem of identifying a profit-maximizing schedule for jobs in a just-in-time production line, where the decision maker can adjust each job's processing time, and thereby its quality level and the profit that can be gained from it. The system comprises two sequential machines with a transition period between them and a setup time after completion of each job on the second machine. We first construct an exact algorithm that maximizes the profit. Since this problem is NP-hard, we construct a fully polynomial time approximation scheme (FPTAS) to address it and evaluate its computational complexity.

**Mathematics Subject Classification.** 65D15, 05Exx, 68Rxx.

Received May 16, 2018. Accepted January 27, 2020.

## 1. Introduction

Production management refers to the application of management principles to the production function in a factory, with the goal of ensuring that the goods or services produced comply with quantitative specifications and a given demand schedule. Production management encompasses all decision-making associated with the production process, including planning, organizing, directing, and control. The current paper focuses on a specific type of production management strategy called just-in-time production, in which items are created to meet demand, instead of being produced in surplus and in advance of need. Just-in-time production aims to reduce the waste associated with overproduction, waiting, and excess inventory. Herein, we seek to determine an optimal schedule for production of a set of items that the manufacturer wishes to complete at specific due dates; in particular, he has the option to complete them either "just in time" or after their due dates (not before), but he does not gain a profit from items that are produced late. We assume that the decision maker can adjust the processing time of each item, and that the choice of a given item's processing time affects the item's quality and, consequently, the profit that the decision maker can gain from the item. It is noteworthy that although the concept of quality is subjective, we suggest that a higher-quality item is likely to take longer to prepare and to be able to command a higher price. Our objective is to identify the production schedule that maximizes the decision maker's profit. We further assume that the production process involves two sequential work stations with a period of transition between them; thus, the problem we address is a permutation flow shop problem.

*Keywords.* Flow shop scheduling, FPTAS, combinatorial optimization, dynamic programming.

Bar Ilan University, Ramat Gan 5290002, Israel.
*Corresponding author: amir.elalouf@biu.ac.il

© EDP Sciences, ROADEF, SMAI 2021

To illustrate the use of the model imagine an aircraft repair workshop. There are a variety of parts that need to be repaired and can be repaired at different levels of quality that will affect the quality of the part and its reliability. The higher the quality level selected, the longer the repair time will be. However, the resources are limited, and each part has a deadline. Decision makers have to determine the schedule and the level of quality for the parts to be repaired.

We first present an exact pseudo-polynomial dynamic programming algorithm to solve the problem. However, as our problem is NP-hard (as elaborated below), the exact solution is applicable only to small instances. To address larger instances in a reasonable computation time, it is necessary to construct a heuristic solution or approximation (see *e.g.* [4,11]). Accordingly, we develop a fully-polynomial time approximation scheme (FPTAS) to address the problem.

Several studies have addressed scheduling problems related to the one presented herein. Miltenburg and Sinnamon [11], for example, used a heuristic technique to solve a mixed model with just-in-time production. Ye *et al.* [17] investigated a complex flow shop problem and constructed a heuristic algorithm to solve it. Perlman *et al.* [13] investigated a production system with two stations, using queuing theory and numerical analysis. Pehrsson *et al.* [12] constructed a method for simultaneous identification of bottlenecks and improvement actions to production systems. Polotski *et al.* [14] found an optimal production scheduling approach for hybrid manufacturing–remanufacturing systems with setups. Elalouf and Wachtel [4] constructed an FPTAS to schedule patient evaluations in an emergency department.

Woeginger [16] presented a natural and uniform approach to fully polynomial time approximation schemes. He indicated that if a combinatorial optimization problem can be formulated through a dynamic program of a particular structure and if the involved expenditure and transition functions please specific arithmetical and structural conditions, then the optimization problem automatically holds a fully polynomial time approximation scheme (FPTAS).

A simple flow shop model that has some similarity to the one we address was investigated by Choi and Yoon [1], who showed that the considered problem is NP-hard. Elalouf *et al.* [3] constructed an FPTAS for that problem. However, in contrast to the current work, Elalouf *et al.* [3] did not address the quality level of the items produced; nor did they consider transportation time between the workstations, or additional setup time required for the production line after an item has been completed (referred to herein as "turnover time").

Another stream of literature related to our study investigates the association between scheduling and quality of service. Mazzeo [10] deals with a practical scheduling decision problem in the U.S airline industry. Daniels and Carrillo [2] deal with scheduling and quality in a system with uncertainty. Toporkov [15] proposes a scheduling approach for achieving a desired level of quality of service in distributed computing.

The rest of this paper is organized as follows: In the next section we introduce the mathematical model of the scheduling problem under consideration and discuss some useful properties of the problem. In Section 3 we design an exact pseudo-polynomial dynamic programming algorithm to solve the problem. In Section 4 we provide a general description of a new FPTAS. In Sections 5–7 we present three sub-procedures for improving the FPTAS' complexity and calculate the computational complexity of the FPTAS. In Section 8 we compare the running times of the exact algorithm (EXACT) and the approximation algorithm (APP). We conclude the paper and suggest future research directions in Section 9.

## 2. Mathematical model

We consider a production line consisting of two workstations, and a set of jobs that need to be completed at specific due dates. Each job has two stages: (a) the *preparation stage*, which is done at the first workstation, and (b) the *operating stage*, which is done at the second workstation; jobs are transported from the first workstation to the second, at a fixed transport time that is the same for all jobs. We further assume that, after completing a job, workstation 2 must undergo a setup procedure ("turnover") prior to accepting the subsequent job. For each job, the decision maker can select the quality level at which the job will be processed in the second stage; each quality level is associated with a different processing time and profit. In particular, we assume that the decision maker can complete jobs either exactly on their due dates or afterward, but that jobs cannot be completed in advance of their due dates. The decision maker gains a profit from a job only if it is completed "just in time" (*i.e.*, no later than its due date). The decision maker's objective is to select a production schedule, and to determine the various jobs' quality levels, so as to maximize his profit.

More formally, let $J$ be a set of $n$ independent non-preemptive jobs $J = \{J_1, \dots, J_n\}$ available at time zero. Let $d_j$ represent the due date of job $J_j$. Each job can be carried out on machine 2 in one of $m$ different modes, each of which is characterized by its own processing time, quality level, and profit. The following parameters are given:

| | |
|---|---|
| $p_{1j}$ | processing time of job $J_j$ on machine 1; |
| $p_{2jh}$ | processing time of job $J_j$ on machine 2 in mode $h, j = 1, \dots, n,\ h = 1, \dots, m$; |
| $w_{jh}$ | the profit from job $J_j$ if mode $h$ is chosen, and provided that the job is completed at $d_j$; |
| $t_j$ | turnover (setup) time after finishing job $J_j$ on machine 2 (the same for all $h$) and |
| $r$ | a fixed transport time between machines 1 and 2. |

We assume that the values of $d_j, w_{jh}, t_i, p_{1j}, p_{2jh}$, and $r$ are each non-negative. The problem is to find the (ordered) subset of jobs that yields the maximal sum of profits, under the constraint that each of the jobs in the subset must be finished exactly at its due date (at $d_j$), not before or after. As noted above, because all the jobs are performed sequentially on the two machines and follow the same route through the machines, this model is called a (permutation) flow shop.

Our problem is a generalization of the classical Knapsack problem, which is NP-hard (see Garey and Johnson [6] for definitions and further details). Our solution approach to this problem is similar to the techniques for treating NP-hard Knapsack-type scheduling problems developed by Gens and Levner [7], Kacem [9], and Zhang *et al.* [18]. Following this approach, we first obtain an exact pseudo-polynomial-time algorithm and then convert it into an FPTAS.

The following claim is evident:

**Proposition 2.1.** *If we have two possible choices $h_1$ and $h_2$ for job $J_j$, the selection of $J_j(h_1)$ dominates $J_j(h_2)$ if $p_{2j}h_1 \leq p_{2j}h_2$ and $w_jh_1 \geq w_jh_2$.*

*After discarding all the dominated operation options, we obtain a sorted list of options for each job, with the entries arranged in increasing order of both parameters $p_{2jh}$ and $w_j$. We assume that each job has at most $m$ possible non-dominated options.*

*A job is called early if it is completed no later than its due date; otherwise, it is tardy. A partition of the set $J$ into two disjointed subsets $E$ (i.e., early) and $T$ (i.e., tardy jobs) is called a feasible schedule if all the jobs belonging to set $E$ are completed on the second machine exactly on their assigned due dates.*

**Proposition 2.2.** *There exists an optimal order on the two workstations (i.e., a schedule) in which all the jobs of $E$ are processed in non-decreasing order of their due dates (i.e., the EDD order) on both machines.*

*Proof.* By contradiction, assume that in the optimal schedule the jobs are not in the EDD order. According to this assumption, we have at least two adjacent jobs belonging to $E$ that are not in the EDD order, *i.e.*, $j_1$ and $j_2$, where $j_2$ is processed immediately after $j_1$, but $d_2 < d_1$. Because $d_2 < d_1$ and the jobs must be finished exactly at $d_j$, we must process $j_2$ before $j_1$ on machine 2. A job is ready for processing on machine 2 after completing its processing on machine 1 plus the fixed $r$ time units for transporting it from machine 1 to machine 2. By swapping $j_1$ and $j_2$ on machine 1, $j_1$ and $j_2$ will still be processed on time and belong to $E$. The swap will put $j_1$ and $j_2$ in the EDD order and will not affect the other jobs in the schedule. This action can be performed on any pair of adjacent jobs not arranged in the EDD order. □

In a feasible schedule, we may assume, without loss of generality, that the jobs in set $E$ are numbered in the EDD order, while the jobs in set $T$ can be processed in an arbitrary order and in any arbitrary mode on both machines after the entire schedule of set $E$ is completed. Since $J_j(h)$ is defined as processing job $j$ in mode $h$, and $w_{jh}$ is the profit obtained from processing $J_j(h)$ and finishing it exactly at $d_j$, our objective is to find a feasible schedule with a maximum weighted number of jobs in set $E$. Thus, our two-stage problem of selecting and scheduling the most urgent operations can be classified in terms of the scheme of Graham *et al.* [8] as a two-machine $n$-job flow shop problem with a non-standard goal of maximizing the weighted number of just-in-time jobs, *i.e.*, $F2|n|\max \Sigma_{\mathrm{Jj(h)}\in E}\, w_{jh}$.

## 3. Dynamic programming algorithm

We first show that our scheduling problem can be solved exactly by applying a dynamic programming (DP) algorithm. Denote $w_{j.\max} = \max_h (w_{jh})$. We prove that our algorithm runs in $O(n^2 W \log m)$ time, where $W = \sum_{j=1}^{n} w_{j.\max}$. Any feasible sub-schedule can be described as a sequence $\pi = (J_{\pi(1)}(h_1), J_{\pi(2)}(h_2), \ldots, J_{\pi(k)}(h_k))$ of jobs $J_{\pi(1)}, J_{\pi(2)}, \ldots, J_{\pi(k)}, k \leq n$, and $h_j \leq m$, all belonging to $E$, *i.e.*, all the $k$jobs in $\pi$ are performed just-in-time. Each feasible sub-schedule $\pi$ can be represented by a quadruple $(\Phi_\pi, D_\pi, L_\pi, B_\pi)$, where $\Phi_\pi$ is the sub-schedule profit defined as follows: $\Phi_\pi = \Sigma_j = \pi(1), \ldots, \pi(k) w_{jh}$; $D_\pi$ is the total processing time on the first machine, *i.e.*, $D = \Sigma_{j=\pi(1),\ldots,\pi(k)} p_{1j}$; $L_\pi$ is the last job in $\pi$, *i.e.*, $L_\pi = J_{\pi(k)}(h_k)$; and $B_\pi$ is the job that is included in $\pi$ prior to $L_\pi$, *i.e.*, $B_\pi = J_{\pi(k-1)}(h_{k-1})$. We keep $B_\pi$ only for backtracking, which is needed for finding the best schedule at the end of the algorithm. To simplify notation, we omit the subscript $\pi$ in the quadruple components: $(\Phi_\pi, D_\pi, L_\pi, B_\pi) = (\Phi, D, L, B)$.

Because job $B = J_{\pi(k-1)}(h_{k-1})$ is the last job to be included in set $E$ in $\pi$ prior to $L = J_{\pi(k)}(h_k)$, the following condition is necessary in order to include job $L = J_{\pi(k)}$ in set $E$ right after job $B = J_{\pi(k-1)}$ and to choose for job $L$ option $h$:

$$\max(D_{\pi(k)} + r, d_{\pi(k-1)} + t_{k-1}) + p_{2,\,\pi(k),h} \leq d_{\pi(k)}. \tag{3.1}$$

**Proposition 3.1.** *For the last job $L = J_{\pi(k)}(h_k)$, which is added to sub-schedule $\pi$, the largest $h_k$ that satisfies condition (3.1) is to be selected.*

*Proof.* According to Proposition 2.1, the choices for a job are sorted entries increasing in both parameters $p_{2jh}$ and $w_{jh}$. The objective is to maximize the sum of $w_{jh}$. We choose the job with the maximum $w_{jh}$ that satisfies condition (3.1). Indeed, for any job $j$ that satisfies this condition, machine 2 will be ready for the next job at the same time $(d_j + t_j)$ for any $h$. $\square$

We use the following definitions. An *extension* of a feasible sub-schedule $\pi$ is a feasible schedule $\pi^{\sim}$ obtained from $\pi$ by adding one or several jobs to set $E$ of $\pi$. A feasible schedule $\pi_1$ *dominates* a feasible schedule $\pi_2$ if, for any extension of $\pi_2$, there exists an extension of $\pi_1$ with the same or a better objective value. Obviously, in this case, $\pi_2$ can be removed from further consideration without loss of optimality. Let $\pi_1$ and $\pi_2$ be two feasible sub-schedules of jobs from the set $\{J_1, \ldots, J_j\}$ with the same last job and the same choice for the last job in $E$: $\pi_1 = (\Phi_1, D_1, L, B_1)$ and $\pi_2 = (\Phi_2, D_2, L, B_2)$. Then, we have the following elimination property:

**Property 3.2.** Sub-schedule $\pi_1$ dominates $\pi_2$ if $D_1 \leq D_2$ and $\Phi_1 \geq \Phi_2$.

*Proof.* Indeed, let $A$ be the set of the jobs that are added after job $j$ to sub-schedule $\pi_2$ in order to create a schedule $\pi_{2A}$. This results in a schedule with an objective value of $\Phi_2 + \Phi(A)$, while $\Phi(A)$ is the additional profit accruing from adding set $A$ to $\pi_2$. Adding set $A$ to set $\pi_1$ (creating a schedule $\pi_{1A}$) can result in the same additional profit $\Phi(A)$, since $D1 \leq D2$. Now about the profit, because $\Phi_{1A} = \Phi_1 + \Phi(A) \geq \Phi_2 + \Phi(A) = \Phi_{2A}$ for any set $A$, sub-schedule $\pi_1$ dominates sub-schedule $\pi_2$. $\square$

**Corollary.** *Consider set $S(i)$ of sub-schedules (quadruples) in which the last job is $J_i$ (wherein each quadruple has a profit $\Phi$ lying between 1 and $W = \sum_{j=1}^{n} w_{j.\max}$). After eliminating all the dominated quadruples, set $S(i)$ contains at most $W$ different quadruples.*

**Lemma 3.3.** *$F2\,|n|\max \Sigma_{Jj\in E} w_j$ can be solved in $O(n^2 W \log m)$ time.*

*Proof.* We use the following forward DP method for enumerating all the non-dominated sub-schedules. We initialize the algorithm by setting $S(0) = \{(0,0,0,0)\}$. Then we sequentially construct sets $S(i)$ of feasible quadruples, where $i = 1,\dots,n$, such that all the quadruples in $S(i)$ have the last job $J_i$. Assume that sets $S(0), S(1),\dots, S(i-1)$ are already constructed. Then, by induction, a current set $S(i)$ is obtained as follows:

– Consider sequentially sets $S(0),\dots,S(i-1)$, one after another, and add a new job $J_i$ to each quadruple $(\Phi, D, L, B)$ in these sets.
– Check for job $j$ if there is a feasible schedule that does not violate the necessary condition.
– Find the largest $p_{2jh}$ that does not violate the necessary condition.
– If adding a new job $J_i(h)$ to quadruple $(\Phi,\ D,\ L,\ B)$ in $S(k)$ (where $k = 0,\dots,i-1$) does not violate inequality (3.1), then a new quadruple $(\Phi + w_{ih}, D + p_{1i}, J_i, L)$ is feasible. A set of such feasible quadruples obtained from a single $S(k)$ by adding job $J_i(h)$ is denoted by $G_k(i)$ (where $k = 0,\dots,i-1$). By the inductive premise, each $S(k)$, where $k = 0,\dots,i-1$, has at most $W$ quadruples; therefore, we need to examine $O(W)$ quadruples from $S(k)$ to obtain each $G_k(i)$, for any fixed $k = 0,\dots,i-1$. Since all $S(k), k = 0,\dots,i-1$, are sorted, all the sets $G_k(i)$ are sorted for any $k$, too.
  Merge all the $G_k(i)$ obtained for different values of $k$ from 0 to $i-1$, one after another, into a single (sorted) set denoted by $T_k(i)$ according to the following rules: $T_0(i) = S(0)$ and $T_k(i) = \text{Merge}((T_{k-1}(i), G_k(i))$. During the merging of each pair $(T_{k-1}(i), G_k(i))$, we eliminate all the dominated quadruples in $T_k(i)$, leaving only $O(W)$ non-dominated quadruples in the resulting set. Denote by $S(i)$ the final set obtained after the merging of the pair $(T_{k-2}(i),\ G_{i-1}(i))$; by construction, this final set has at most $W$ (non-dominated) quadruples.

Because only dominated schedules are eliminated during the above procedure, at the end of the considered algorithm, we obtain the required sets $S(1)$ to $S(n)$ that contain an optimal solution. In other words, the optimal solution is a quadruple with the maximum profit among all the quadruples in sets $S(1)$ to $S(n)$.

We formally present the optimization algorithm, called Algorithm 1, in Figure 1.

**Algorithm complexity**: since we discard the dominated quadruples, there are in total at most $W$ quadruples in any of the sets $G_k$(i), $T(i)$, and $S(i)$, for any $i$ and $k$. Furthermore, constructing each $G_{\mathrm{k}}(i)$ in lines 9–16 requires $O(W \log m)$ elementary operations, since each $G_k(i)$ is constructed from a single set $S(i)$. The log $m$ part seeks to search for the best feasible choice in a sorted list with at most $m$ choices (line 13). Merging the sorted sets $G_k(i)$ and $T_{k-1}(i)$, as well as discarding all the dominated quadruples (line 17), is done in linear (in the number of quadruples) time. Since $G_k$(i) and $T_{k-1}(i)$ contain at most $W$ quadruples each, the operations in line 12 are done in $O(W)$ time. In lines 5–20, we have two nested loops, the first starting in line 6 and the second in line 8. Each loop is done in $O(n)$ iterations; in total we have to do $O(n^2)$ iterations. Thus, we have $O(n^2 W \log m)$ elementary operations in total. □

## 4. The FPTAS

We construct an FPTAS for the considered problem in three stages.

**Stage A:** let UB be an upper bound on the total profit; on the basis of the discussion above,

$$\text{UB} = W = \sum_{j=1}^{n} w_{j.\max}.$$

Find a lower bound LB on the total profit such that UB/LB $\le n$. Notice that $\text{LB} = \max_j (w_{j.\max})$ because any single job $J_j$ can be considered as an $E$-component of the feasible schedule $(E = \{J_i\};\ T = (J \backslash \{J_i\})$.

**Stage B:** improve the UB/LB ratio to UB/LB $\le 2$ using the algorithm BOUNDS-M, presented in Section 7. (BOUNDS-M is a modification of an algorithm called **BOUNDS**, presented in Sect. 6; BOUNDS improves the ratio to UB/LB $\le 4$. Both BOUNDS and BOUNDS-M incorporate a sub-algorithm called Test, presented in Sect. 5.)

**Algorithm 1**. An exact pseudo-polynomial DP algorithm

Number all the jobs in increasing order of their due dates (the EDD order). All the choices in each job are sorted in increasing order of $p_{2jh}$ (and $w_{jh}$ after eliminating the dominated choices).

```
1.  Input: $r$, $n$, $m$ ($j = 1, \ldots, n$ , $h = 1, \ldots, m$)
2.  Input for each job $Jj$: $d_j$, $p_{1j}$, $p_{2jh}$, $w_{jh}$
3.  Output: Maximum-profit JIT schedule.
4.  **Step 1**. [Initialization]. Set $S(0) = \{(\Phi = 0, D = 0, L = 0, B = 0)\}$
5.  **Step 2**. [Generate $S(1)$ to $S(n)$]
6.  for $i = 1$ to $n$ do
7.    $T_0(i) \leftarrow \varnothing$
8.    for $k = 1$ to $i$-1 do
9.      $G_k(i) \leftarrow \varnothing$
10.     for each quadruple ($\Phi, D, L = k, B = J_{prior(k)}$) in $S(k)$ do
11.       If there is at least one choice that satisfies:
12.       $\max(D_{\pi(k)} + r, d_{\pi(k-1)} + \mathrm{t}_{\mathrm{k-1}}) + p_{2,\pi(k),h} \le d_{\pi(k)}$, then
13.         Find the choice $h$ with the maximum benefit $w_{ih}$ that satisfies the condition.
14.         Update $G$: $G_k(i) \leftarrow G_k(i) \cup (\Phi + w_{ih}, D + p_{1i}, i, k)$
15.       EndIf
16.     End for
17.     $T_k(i) \leftarrow \mathrm{merge}(T_{k-1}(i), G_k(i))$; during merging eliminate the dominated quadruples
18. End for
19. $S(i) \leftarrow T_{i-1}(i)$
20. End for
21. **Step 3**. Find the maximum-profit quadruple in $S(1), S(2), \ldots, S(n)$.
22. Obtain the optimal set $E$ of JIT jobs by backtracking.
```

FIGURE 1. The exact pseudo-polynomial DP algorithm (JIT: just-in-time).

**Stage C:** find an $\varepsilon$-approximation solution using algorithm AA(LB, UB, $\varepsilon$) presented below; return the $\varepsilon$-approximate schedule, *i.e.*, set $E$ and value $\Phi_{\text{appr}}$.

First, we describe a new $\varepsilon$-approximation algorithm, which, given any instance of the considered just-in-time scheduling problem and an allowable error $\varepsilon > 0$, returns a solution $\pi_{\text{appr}}$ whose profit $\Phi_{\text{appr}} = \Phi(\pi_{\text{appr}})$ is within $\varepsilon\Phi^*$ from the unknown optimal value $\Phi^*$ for the problem instance being considered. More precisely, $\Phi_{\text{appr}} = \Phi(\pi_{\text{appr}}) \geq (1-\varepsilon)\Phi^*$.

Before presenting the algorithm, we begin with the following observations:

– All of the jobs are numbered in increasing order of their due dates (the EDD order).
– Associated with each schedule $\pi$ is a quadruple ($\Phi$, $D$, $L$, $B$), as introduced in Section 3. As in Algorithm 1, the quadruples are arranged in increasing order of the $\Phi$ values. In order to restore an $\varepsilon$-approximation solution corresponding to any quadruple, we use the standard backtracking operation that defines a predecessor job $B$ for the last job in any quadruple.
– If there are two quadruples $(\Phi_1, D_1, L_1, B_1)$ and $(\Phi_2, D_2, L_2, B_2)$ such that $\Phi_1 \geq \Phi_2$ and $D_1 \leq D_2$, then the quadruple $(\Phi_2, D_2, L_2, B_2)$ is dominated and should be discarded.
– Let $\delta = \varepsilon\text{LB}/n$. If there are two quadruples in set $S(i)$ (which has the last job $J_i$ in all its quadruples, as defined in Sect. 3) such that $0 \leq \Phi_2 - \Phi_1 \leq \delta$, then the quadruples are called $\delta$-close. We use the operation called *discarding* $\delta$*-close quadruples* from set $S(i)$, which means the following:
  (a) Partition the interval [0, UB] into $\ulcorner(\text{UB}/\text{LB})(n/\varepsilon)\urcorner$ subintervals of equal size no greater than $\delta = \varepsilon\text{LB}/n$;
  (b) If more than one quadruple from $S(i)$ falls into any one of the above subintervals, then discard all such $\delta$-close quadruples, leaving only one *representative* quadruple in each subinterval, namely, the one with the smallest (in this sub-interval) $D$-coordinate.

Now we are ready to present a new $\varepsilon$-approximation algorithm based on Algorithm 1 (see Fig. 2).

**Property 4.1.** The complexity of AA(LB, UB, $\boldsymbol{\varepsilon}$) is $O(n^3(\log m)(\text{UB}/\text{LB})(1/\varepsilon))$.

*Proof.* Since the sub-interval length is $\delta = \varepsilon\text{LB}/n$, we have $O(n(\text{UB}/\text{LB})(1/\varepsilon))$ sub-intervals in the interval $[0, \text{UB}]$. Because there is at most one representative quadruple in each sub-interval, we have in total $O(n(\text{UB}/\text{LB})(1/\varepsilon))$ quadruples in any of the sets $G_k$(i), $T(i)$, and $S(i)$, for any $i$ and $k$. Furthermore, constructing each $G_{\text{k}}(i)$ in lines 9–16 in Figure 2 requires $O(n(\text{UB}/\text{LB})(1/\varepsilon)(\log m))$ elementary operations. The log $m$ part corresponds to a search for the best feasible choice in a sorted list with at most $m$ choices (line 13). Merging the sorted sets $G_k(i)$ and $T_{k-1}(i)$, as well as discarding all the dominated and $\delta$-close quadruples (line 17), is done in linear (in the number of quadruples) time. Since $G_k$(i) and $T_{k-1}(i)$ contain at most $n(\text{UB}/\text{LB})(1/\varepsilon)$ quadruples each, the operations in line 17 are done in $O(n(\text{UB}/\text{LB})(1/\varepsilon))$ time. In lines 5–20, we have two nested loops, the first starting in line 6 and the second in line 8. Each loop is done in $O(n)$ iterations; in total we have to do $O(n^2)$ iterations. Thus, we have $O(n^3(\log m)(\text{UB}/\text{LB})(1/\varepsilon))$ elementary operations in total. □

As long as the ratio UB/LB $\leq n$, the algorithm AA(LB, UB, $\varepsilon$) runs in $O(n^4(1/\varepsilon)(\log m))$ time. We can improve the complexity by a factor of $n$. To do this, we construct a procedure called BOUNDS (Sect. 4), which finds an improved ratio UB/LB $\leq 4$, which runs in $O((\log m)n^3 \log\log n)$. Next, we further improve that procedure by developing a modified procedure called BOUNDS-M, which obtains a ratio of UB/LB $\leq 2$, and runs faster than BOUNDS, in $O((\log m)n^3)$. BOUNDS and BOUNDS-M are based on a parametric sub-algorithm called Test($v$, $\varepsilon$). We elaborate on each of those procedures in the following sections.

## 5. Testing procedure: a sub-algorithm for bounds and BOUNDS-M

The testing procedure Test($v$, $\varepsilon$) is applied repeatedly as a sub-procedure in algorithms BOUNDS and BOUNDS-M to narrow the gap between UB and LB until UB/LB $\leq 4$ and UB/LB $\leq 2$, respectively. This sub-algorithm has the following property: if it outputs "yes", then the maximum weighted number of just-in-time jobs $\Phi^*$ is definitely larger than or equal to $v$, *i.e.*, $\Phi^* \geq v$; if it outputs "no", then $\Phi^* \leq v(1+\varepsilon)$.

**Algorithm 2**. An ε -approximation algorithm AA($LB$, $UB$, ε)

Number all the jobs in increasing order of their due dates (the EDD order). All the possibilities in each job are sorted in increasing order of $p_{2jh}$ (and $w_{jh}$ after eliminating the dominated possibilities).

```
1.  Input: $r, n, m$ ($j = 1, \dots, n$ , $h = 1, \dots, m$)
2.  Input for each job $Jj$: $d_j, p_{1j}, p_{2jh}, w_{jh}$
3.  Output: An ε-approximate schedule.
4.  Step 1. [Initialization]. Set $S(0) = \{(\Phi = 0, D = 0, L = 0, B = 0)\}$, $\delta \leftarrow \varepsilon LB/n$
5.  Step 2. [Generate $S(1)$ to $S(n)$]
6.  for $i = 1$ to $n$ do
7.    $T_0(i) \leftarrow \varnothing$
8.    for $k = 1$ to $i$-1 do
9.      $G_k(i) \leftarrow \varnothing$
10.     for each quadruple ($\Phi$, $D$, $L = k$, $B = J_{prior(k)}$) in $S(k)$ do
11.       If there is at least one choice that satisfies:
12.       $\max(D+\mathrm{r}, d_{\pi(k-1)}+\mathrm{t}_{k-1}) + p_{2,\pi(k),h} \le d_{\pi(k)}$, then
13.         Find the choice $h$ with the maximum benefit $w_{ih}$ that satisfies the condition.
14.         Update $G$: $G_k(i) \leftarrow G_k(i) \cup (\Phi+w_{ih}, D+p_{1i}, i, k)$
15.       EndIf
16.     End for
17.     $T_k(i) \leftarrow$ merge($T_{k-1}(i)$,$G_k(i)$); during merging discard the dominated quadruples
        and $\delta$-close quadruples
18. End for
19. $S(i) \leftarrow T_{i-1}(i)$
20. End for
21. Step 3. Find the maximum-profit quadruple in $S(1), S(2), \dots, S(n)$.
22. The latter profit is $\Phi(\pi_{\mathrm{appr}})$, which is at least $(1-\varepsilon)\Phi^*$, where $\Phi^*$ is the optimal profit.
23. Obtain the ε-optimal solution by backtracking.
```

FIGURE 2. The $\varepsilon$ -approximation algorithm AA(LB,UB, $\varepsilon$).

Test($v$, $\varepsilon$) works similarly to the $\varepsilon$-approximation algorithm AA(LB, UB, $\varepsilon$) (Algorithm 2), but uses a different value of $\delta$ and has a different form of output. A full description of Test is presented in Figure 3. In particular, Test($v$, $\varepsilon$) has the following features:

(i) Whereas in AA(LB, UB, $\varepsilon$) we partition the interval [0, UB] into $\lceil(\mathrm{UB}/\mathrm{LB})\,(n/\varepsilon\rceil$ subintervals, in Test($v$, $\varepsilon$) we partition the interval [0, $v$] into $\lceil(n/\varepsilon\rceil$ subintervals.

(ii) Whereas AA(LB, UB, $\varepsilon$) continues its work until it performs all the iterations in the loops, Test($v$, $\varepsilon$) can terminate when it finds a feasible schedule with a profit greater than or equal to $v$.

(iii) Whereas AA(LB, UB, $\varepsilon$) returns an $\varepsilon$-approximation set, Test($v$, $\varepsilon$) depends on the parameter and returns "yes", if $\Phi^* \geq v$ or "no" if $\Phi* \leq (1+\varepsilon)v$.

(iv) If Test($v, \varepsilon$) outputs "yes", then the maximum weighted number of just-in-time jobs $\Phi^*$ is larger than or equal to $v$; if it outputs "no", then $\Phi^* \leq v(1+\varepsilon)$. The proof of this claim is along the same lines as the proofs of the test validity in Gens and Levner [7], and is omitted here.

**Property 5.1.** The complexity of Test($v$, $\varepsilon$) is $O(n^3(1/\varepsilon)(\log\, m))$.

*Proof.* Since the sub-interval length is $\delta = \varepsilon v/n$, we have $O(n/\varepsilon)$ sub-intervals in the interval [0, $v$]. Because there is at most one representative quadruple in each sub-interval, we have in total $O(n/\varepsilon)$ quadruples in any of the sets $G_k$(i), $T(i)$, and $S(i)$, for any $i$ and $k$. Furthermore, constructing each $G_{\mathrm{k}}(i)$ in lines 9–16 requires $O(n/\varepsilon)$ elementary operations. The log $m$ part corresponds to a search for the best feasible choice in a sorted list with at most $m$ choices (line 13). Merging the sorted sets $G_k(i)$ and $T_{k-1}(i)$, as well as discarding all the dominated and $\delta$-close quadruples (line 18), is done in linear (in the number of quadruples) time. Since $G_k$(i) and $T_{k-1}(i)$ contain at most $O(n/\varepsilon)$ quadruples each, the operations in line 18 are done in $O(n/\varepsilon)$ time. In lines 5–20, we have two nested loops, the first starting in line 6 and the second in line 8. Each loop is done in $O(n)$ iterations; in total we have to do $O(n^2)$ iterations. Thus, we have $O(n^3(1/\varepsilon)(\log\, m))$ elementary operations in total. ☐

## 6. The narrowing procedure: BOUNDS

Using Test($v, \varepsilon$), we can obtain the BOUNDS algorithm presented in Figure 4, which improves the UB/LB ratio to UB/LB $\leq 4$.

In what follows, Test $(v, 1)$ denotes Test($v, \varepsilon$) with fixed $\varepsilon = 1$. Clearly, if Test ($v$, 1) outputs "yes", then $\Phi \geq v$; if it outputs "no", then $\Phi^* \leq 2v$.

In the algorithm description (Fig. 4), $T_{\mathrm{UB}}$ denotes a tentative upper bound (note that it is allowed to be smaller than the optimal value $\Phi^*$ in some steps of the algorithm).

**Property 6.1.** The LB and UB returned by **BOUNDS** satisfy the following conditions:

(1) LB $\leq \Phi^* \leq$ UB,
(2) UB/LB $\leq 4$.

*Proof.* The initial LB and UB values (the input) satisfy the first condition. While the initial UB must satisfy UB $\geq \Phi^*$, $T_{\mathrm{UB}}$ in line 5 satisfies a relaxed condition $T_{\mathrm{UB}} \geq 0.5\ \Phi^*$. Lines 2–6 are the logarithmic binary search loop. In each iteration, we test a new $v$ value and use it to update LB or $T_{\mathrm{UB}}$.

Consider two cases:

Case 1. If Test($v$, 1) in line 5 returns "yes", then $\Phi^* \geq v$. Thus LB is increased to $v$. In this case, $\Phi^* \geq v$ and LB remains $\leq \Phi^*$ every time when the algorithm performs this update.

Case 2. If Test($v$, 1) in line 5 returns "no", then $\Phi^* \leq 2v$ and $T_{\mathrm{UB}}$ is updated to $v$. In this case, $v \geq 0.5\Phi^*$, *i.e.*, each time when we update $T_{\mathrm{UB}}$ it will be at least $0.5\Phi^*$. The loop in lines 2–6 runs until $T_{\mathrm{UB}}$/LB $\leq 2$. When the loop terminates, UB is updated to $2T_{\mathrm{UB}}$ (line 7). After this operation, $\Phi^*$ must lie between the (new) LB and the (new) UB. Since $T_{\mathrm{UB}}$/LB $\leq 2$ and UB $= 2T_{\mathrm{UB}}$, when the algorithm terminates UB/LB $\leq 4$. ☐

**Algorithm 3**. Test ($v$, ε)

Number all the jobs in increasing order of due dates (the EDD order). All the choices in each job are sorted in increasing order of $p_{2jh}$ (and $w_{jh}$ after eliminating dominated choices).

```
1. Input: $r$, $n$, $m$ (j= 1, …, $n$ , $h$ = 1, …, $m$)
2. Input for each job $Jj$: $d_j$, $p_{1j}$, $p_{2jh}$, $w_{jh}$
3. Output: Report “yes” if $\Phi^* \geq v$ and “no” if $\Phi^* \leq (1+\varepsilon)v$
4. Step 1. [Initialization]. Set $S(0) = \{(\Phi = 0, D = 0, L = 0, B = 0)\}$, $\delta \leftarrow \varepsilon v/n$
5. Step 2. [Generate $S(1)$ to $S(n)$]
6. for $i = 1$ to $n$ do
7.   $T_0(i) \leftarrow \varnothing$
8.   for $k = 1$ to $i$-1 do
9.     $G_k(i) \leftarrow \varnothing$
10.    for each quadruple ($\Phi$, $D$, $L = k$, $B = J_{prior(k)}$) in $S(k)$ do
11.      If there is at least one choice that satisfies:
12.      $\max(D+r, d_{\pi(k-1)}+t_{k-1}) + p_{2,\pi(k),h} \leq d_{\pi(k)}$, then
13.        Find the choice $h$ with the maximum benefit $w_{ih}$ that satisfies the condition.
14.        Update $G$: $G_k(i) \leftarrow G_k(i) \cup (\Phi+w_{ih}, D+p_{1i}, i, k)$
15.        If (in the new quadruple) $\Phi \geq v$, then go to 24
16.      EndIf
17.    End for
18.    $T_k(i) \leftarrow$merge($T_{k-1}(i)$, $G_k(i)$); during merging eliminate the dominated quadruples and
       $\Delta$-close quadruples
19. End for
20. $S(i) \leftarrow T_{i-1}(i)$
21. End for
22. Step 3. The answer
23. Return “no” (i.e., $\Phi^* \leq (1+\varepsilon)v$) and End.
24. Return “yes” (i.e., $\Phi^* \geq v$).
```

FIGURE 3. The testing procedure Test($v$, $\varepsilon$).

**Algorithm 4. BOUNDS**

```
Input: $LB$ and $UB$ such that $UB/LB \leq n$.
Output: $LB$ and $UB$ such that $UB/LB \leq 4$
    1. Set $T_{UB} \leftarrow UB$
    2. If $T_{UB}/LB \leq 2$ then go to 7
    3. Set $v \leftarrow \sqrt{LB \cdot T_{UB}}$
    4. Run Test($v$, 1)
    5. If Test($v$, 1) = "yes", set $LB \leftarrow v$, else $T_{UB} \leftarrow v$
    6. Go to 1
    7. $UB = 2T_{UB}$
    8. End
```

FIGURE 4. BOUNDS algorithm.

**Property 6.2.** The complexity of BOUNDS is $O((\log\, m)n^3 \log\log n)$.

*Proof.* The logarithmic binary search that reduces the ratio UB/LB from $n$ to 4 requires log log $n$ iterations. Test($v$, 1) runs in $O((\log\, m)n^3)$ time and does not depend on $\varepsilon$. Thus, the total complexity of BOUNDS is $O(n^3 \log\, \log\, n)$. □

## 7. MODIFIED NARROWING PROCEDURE: BOUNDS-M

In this section we present a modification of BOUNDS that originates from the procedure suggested by Ergun *et al.* [5] for solving the restricted shortest path problem. Although we are solving a different problem, we use the key idea presented by Ergun *et al.* [5]; namely, when we run Test$(v, \varepsilon)$, we choose $\varepsilon$ to be a function of UB/LB, changing from iteration to iteration. For the reader's convenience, we distinguish the allowable error ($\varepsilon$) in the FPTAS from the iteratively-changing error in the testing procedure by denoting the latter by $\theta$. Accordingly, the testing procedure presented in Section 5 will subsequently be referred to as Test$(v, \theta)$. The idea is that when UB and LB are far from each other, we choose a large $\theta$; when UB and LB get closer, we choose a smaller $\theta$. More precisely, similarly to Ergun *et al.* [5], in each iteration of Test$(v, \theta)$, we set $\theta = \sqrt{\text{UB}/\text{LB}} - 1$, whereas $v$ takes the value of $v = \sqrt{\text{LB} \cdot \text{UB}/\,(1+\theta)}$.

Although the modified algorithm BOUNDS-M for the scheduling problem literally coincides with the corresponding narrowing algorithm of Ergun *et al.* [5] for the routing problem, we present it in Figure 5 for the sake of completeness.

The complexity of BOUNDS-M is $O((\log\, m)n^3)$. The proof is just the same as that of Lemma 5 in Ergun *et al.* [5]. The only difference is that the complexity of Test$(v, \theta)$ in the routing problem of Ergun *et al.* [5] is $O(mn/\theta)$, whereas in our scheduling problem the complexity of Test$(v, \theta)$ is $O((\log\, m)n^3/\theta)$.

Furthermore, the computational complexity of the FPTAS ought to be calculated.

**Theorem 7.1.** *The time complexity of the FPTAS is* $O((\log\, m)n^3/\varepsilon)$.

**Algorithm 5. BOUNDS-M**

*Input*: *LB* and *UB* such that $UB/LB \leq n$.
*Output*: *LB* and *UB* such that $UB/LB \leq 2$

```
1. If $UB/LB \leq 2$ then go to 7
2. Set $\theta = \sqrt{UB/LB} - 1$
3. Set $v \leftarrow \sqrt{LB \cdot UB/(1+\theta)}$
4. Run Test($v$, $\theta$)
5. If Test($v$, $\theta$) = “yes” then set $LB \leftarrow v$, else $UB \leftarrow v(1+\theta)$
6. Go to 1
7. End
```

FIGURE 5. BOUNDS-M algorithm.

*Proof.* Stage A: at this stage, we find $w_{\max}$ and $W$. For this purpose, a single scan over all values of $w_{jh}$ is needed, which is carried out in $O(n \log m)$ time.

Stage B: the procedure BOUNDS-M is performed in $O((\log m)n^3)$ time.

Stage C: at this stage, AA(LB, UB, $\varepsilon$) runs in $O((\log m)n^3/\varepsilon)$.

Therefore, the total complexity of the FPTAS is $O((\log m)n^3/\varepsilon)$. □

## 8. Computational experiments

In order to examine the computational properties of the new approximation algorithm, we compared, for several scenarios, the running times of the proposed approximation algorithm (APP) with the exact algorithm (EXACT).

We coded all the programs in Java and ran them on a computer with Intel Core i5 CPU, 3.06 GHz, 8 GB memory. Table 1 exemplifies the testing results. It illustrates the impact of the attribute sizes on the running time; parameter $w$ denotes the maximum value of $w_{jh}$ considered. From the experiment results, one can observe that the pseudo-polynomial running time depends on UB (see Lem. 3.3), whereas UB depends on the $w_{jh}$ values. The table also vividly demonstrates that the average running time of the approximation algorithm APP depends on different $\varepsilon$-values. Figures 6(Graph 1) and 7(Graph 2) display the comparison results. While Figure 6(Graph 1) compares the running time of different scenarios of EXACT, Figure 7(Graph 2) compares the running time of APP and EXACT with epsilon = 0.01.

The entries with an asterisk required more than one hour of the average running time in our experiments. The values were obtained by extrapolation of the experimental data on the basis of theoretical worst-case estimations.

## 9. Conclusion and future work

We have considered the problem of scheduling a just-in-time production process, in which the decision maker can select the jobs’ quality levels, thereby determining their processing times and associated profits.

TABLE 1. The average running time of the exact algorithm and the approximation algorithm.

| Network size | | Average running time (s) of the exact algorithm | | Average running time (s) of the approximation algorithm | |
|---|---|---|---|---|---|
| # of jobs | # of modes | $w$ =1000 | $w$ =2500 | $\varepsilon = 0.1$ | $\varepsilon = 0.01$ |
| 20 | 100 | Less than 1 s | Less than 1 s | Less than 1 s | Less than 1 s |
| 100 | 100 | 6 | 17 | Less than 1 s | Less than 1 s |
| 200 | 200 | 53 | 164 | Less than 1 s | 6 |
| 500 | 200 | 430 | 1008 | 9 | 96 |
| 700 | 400 | 1553 | 3595 | 32 | 296 |
| 1000 | 400 | 3183 | 8317* | 92 | 864 |
| 1200 | 500 | 7613* | 14 886* | 155 | 1549 |

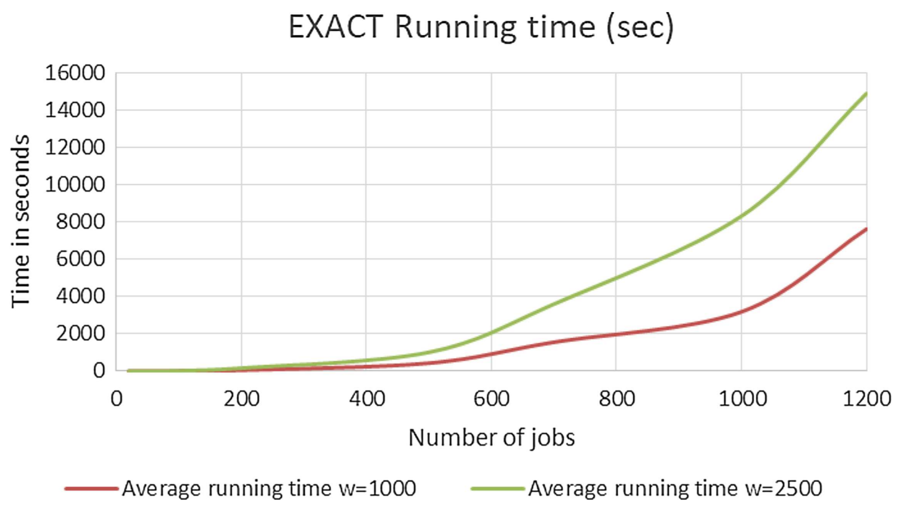

FIGURE 6. Graph 1. EXACT Running time.

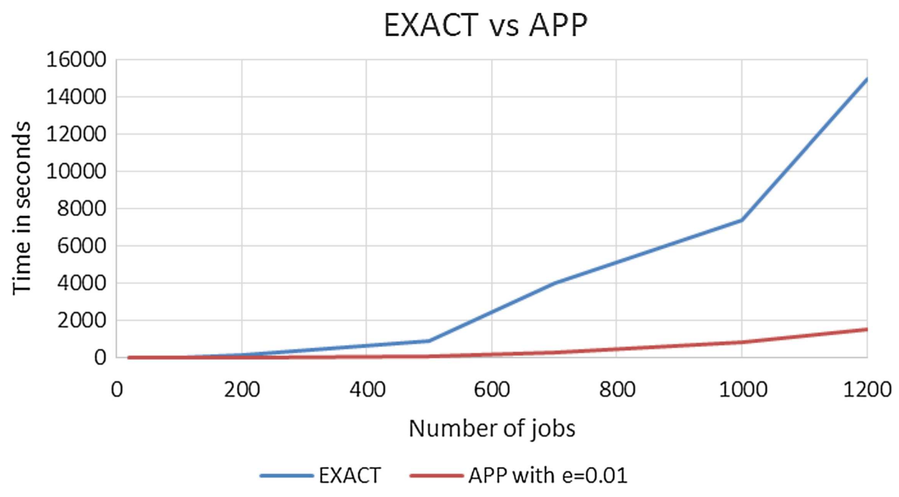

FIGURE 7. Graph 2. Running time of EXACT vs APP.

The suggested scheduling model provides a basis for the design of fast practical algorithms. We developed a new dynamic programming (DP) algorithm to solve the problem in pseudo-polynomial time. Using the DP algorithm, we constructed a new FPTAS for the problem.

Whether or not the running time of the FPTAS can be further improved remains an interesting open question. Future work should be undertaken in the following directions:

– A multi-stage modification of the suggested two-stage scheduling model that permits the analysis and treatment of more complicated production systems;
– Design of on-line algorithms and experimental evaluation of their performance in practice;
– An experimental evaluation of various scheduling scenarios including dynamic, learning and deterioration factors;
– Present-day advances in IT technologies, such as geographical information systems, mobile agents, and RFIDs, should be introduced into the scheduling model.

## References

[1] B.C. Choi and S.H. Yoon, Maximizing the weighted number of just-in-time jobs in flow shop scheduling. *J. Sched.* **10** (2007) 237–243.

[2] R.L. Daniels and J.E. Carrillo, $\beta$-Robust scheduling for single-machine systems with uncertain processing times. *IIE Trans.* **29** (1997) 977–985.

[3] A. Elalouf, E. Levner and H. Tang. An improved FPTAS for maximizing the weighted number of just-in-time jobs in a two-machine flow shop problem. *J. Sched.* **16** (2013) 429–435.

[4] A. Elalouf and G. Wachtel, An alternative scheduling approach for improving emergency department performance. *Int. J. Prod. Econ.* **178** (2016) 65–71.

[5] F. Ergun, R. Sinha and L. Zhang, An improved FPTAS for restricted shortest path. *Inf. Process. Lett.* **83** (2002) 287–291.

[6] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co, New York, NY (1979).

[7] G.V. Gens and E.V. Levner, Fast approximation algorithm for job sequencing with deadlines. *Discrete Appl. Math.* **3** (1981), 313–318.

[8] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* **5** (1979), 287–326.

[9] I. Kacem, Fully polynomial-time approximation scheme for the weighted total tardiness minimization with a common due date. *Discrete Appl. Math.* **158** (2010) 1035–1040.

[10] M.J. Mazzeo, Competition and service quality in the US airline industry. *Rev. Ind. Organiz.* **22** (2003) 275–296.

[11] J. Miltenburg and G. Sinnamon, Scheduling mixed-model multi-level just-in-time production systems. *Int. J. Prod. Res.* **27** (2007) 1487–1509.

[12] L. Pehrsson, A.H.C. Ng and J. Bernedixen, Automatic identification of constraints and improvement actions in production systems using multi-objective optimization and post-optimality analysis. *J. Manuf. Syst.* **39** (2016), 24–37.

[13] Y. Perlman, A. Elalouf and E. Bodinger, Dynamic repair priority for a transfer line with a finite buffer. *J. Manuf. Syst.* **33** (2014) 16–26.

[14] V. Polotski, J. Kenne and A. Gharbi, Optimal production scheduling for hybrid manufacturing–remanufacturing systems with setups. *J. Manuf. Syst.* **37** (2015) 703–714.

[15] V. Toporkov, Application-level and job-flow scheduling: an approach for achieving quality of service in distributed computing. In: International Conference on Parallel Computing Technologies. Springer, Berlin Heidelberg (2009) 350–359.

[16] G.J. Woeginger, When does a dynamic programming formulation guarantee the existence of an FPTAS? *INFORMS J. Comput.* **12** (2000) 57–74.

[17] H. Ye, W. Li and E. Miao, An effective heuristic for no-wait flow shop production to minimize makespan. *J. Manuf. Syst.* **40** (2016) 2–7.

[18] L.Q. Zhang, L.F. Lu and C.T. Ng, The unbounded parallel-batch scheduling with rejection. *J. Oper. Res. Soc.* **63** (2012) 293–298.