No-idle, no-wait: when shop scheduling meets dominoes, eulerian and hamiltonian paths

J-C. Billaut[a], F. Della Croce[b,c], F. Salassa[b], V. T'kindt[a]

[a] *Université Francois-Rabelais de Tours, ERL CNRS OC 6305, Tours, France*
[b] *DIGEP, Politecnico di Torino, Corso Duca degli Abruzzi 24, Torino, Italy*
[c] *CNR, IEIIT, Torino, Italy*

Abstract

In shop scheduling, several applications exist where it is required that some components perform consecutively. We refer to no-idle schedules if machines are required to operate with no inserted idle time and no-wait schedules if tasks cannot wait between the end of an operation and the start of the following one. We consider here no-idle/no-wait shop scheduling problems with makespan as performance measure and determine related complexity results. We first analyze the two-machine no-idle/no-wait flow shop problem and show that it is equivalent to a special version of the game of dominoes which is polynomially solvable by tackling an Eulerian path problem on a directed graph. We present for this problem an O(n) exact algorithm. As a byproduct we show that the Hamiltonian Path problem on a digraph G(V,A) with a special structure (where every pair of vertices i,j either has all successors in common or has no common successors) reduces to the two-machine no-idle/no-wait flow shop problem. Correspondingly, we provide a new polynomially solvable special case of the Hamiltonian Path problem. Then, we show that also the corresponding m-machine no-idle no-wait flow shop problem is polynomially solvable and provide an $O(mn \log n)$ exact algorithm. Finally we prove that the 2-machine no-idle/no-wait job shop problem and the 2-machine no-idle/no-wait open shop problem are NP-Hard in the strong sense.

Keywords: No-idle no-wait shop scheduling, dominoes, eulerian path, hamiltonian path, Numerical Matching with Target Sums

1. Introduction

In shop scheduling, typically when machines represent very expensive equipments and the fee is directly linked to the actual time consumption, it

is of interest to determine so-called no-idle solutions, that is schedules where machines process the jobs continuously without inserted idle time. Also, particularly in metal-processing industries, where delays between operations interfere with the technological process, it is required to obtain no-wait schedules where each job is subject to the so-called no-wait constraint, that is it cannot be idle between the completion of an operation and the start of the following one.

We attack here the simultaneous combination of these two requirements by considering no-idle/no-wait shop scheduling problems with three different shop configurations namely flow shop, job shop and open shop. We focus on the makespan as performance measure. We first deal with the two-machine flow shop problem and the m-machine flow shop problem showing that both are polynomially solvable and present connections with the game of dominoes and well-known graph problems. Then, we prove that the two-machine job shop and the two-machine open shop are NP-hard in the strong sense. Using the standard three-field notation [16], the three two-machine shop problems are denoted as $F2|no{-}idle, no{-}wait|C_{\max}$ for the flow shop, $J2|no{-}idle, no{-}wait|C_{\max}$ for the job shop and $O2|no-idle, no-wait|C_{\max}$ for the open shop, respectively, while the general flow shop case is denoted as $F|no-idle, no-wait|C_{\max}$.

With respect to the relevant literature, in one of the pioneering works in scheduling [14], it is shown that problem $F2||C_{\max}$ is solvable in $O(n \log n)$ time by first arranging the jobs with $p_{1,j} \leq p_{2,j}$ in non-decreasing order of $p_{1,j}$, followed by the remaining jobs arranged in non-increasing order of $p_{2,j}$, where $p_{i,j}$ denotes the processing time of job J_j on machine M_i. As mentioned in [1], $F2|no{-}idle|C_{\max}$ can also be solved in $O(n \log n)$ time by simply packing the jobs on the second machine once the schedule computed by the algorithm in [14] is given. In [17], it is shown that problem $F2|no{-}wait|C_{\max}$ can be seen as a special case of the Gilmore-Gomory Travelling Salesman Problem [10] and therefore is solvable too in $O(n \log n)$ time. Besides, problems $F3||C_{\max}$, $F3|no-idle|C_{\max}$ and $F3|no-wait|C_{\max}$ were all shown to be NP-hard in the strong sense by [7], [3] and [18] respectively. In [1], it is reported that both problems $F2|no-idle|\sum C_j$ and $F2|no-wait|\sum C_j$ are NP-hard by exploiting the fact that the NP-hardness proof of problem $F2||\sum C_j$ in [7] was given by constructing a flow shop instance that happened to be both no-idle and no-wait. Similar consideration holds for problem $F2|no-idle, no{-}wait|\sum C_j$. For surveys on $no{-}idle$ flow shop scheduling, we refer to [11, 4]. For surveys on no-wait scheduling, we refer to [12, 2]. In [15], the links between problems $F|no{-}idle|C_{\max}$ and $F2|no{-}wait|C_{\max}$ are discussed

and some efficiently solvable special cases are shown. The recent literature on $no-wait$ flow shop scheduling includes [13] where it is shown that minimizing the number of interruptions on the last machine is solvable in $O(n^2)$ time on two machines (the problem is denoted as $F2|no-wait|\mathcal{G}$) while it is NP-hard on three or more machines. Finally, we mention the contribution of [9], where it is shown that, if some processing times are allowed to be zero and zero processing times imply that the corresponding operations should not be performed, then problem $F2|no-idle, no-wait|C_{\max}$ is NP-hard in the strong sense. Here, we deal with the standard versions of no-idle/no-wait shop problems where all processing times are required to be strictly positive.

The paper proceeds as follows. In Section 2, we show that problem $F2|no-idle, no-wait|C_{\max}$ is equivalent to an oriented version of the Single Player Dominoes problem which has been shown in [5] to be polynomially solvable and presents an $O(n)$ time solution approach. As a byproduct, we also consider a special case of the Hamiltonian Path problem (denoted as Common/Distinct Successors Directed Hamiltonian Path - CDSDHP - problem) on a directed graph $G(V, A)$ with a specific structure so that every pair of vertices i, j either has all successors in common or has no common successor. In other words, either the successors of i coincide with the successors of j or i, j have distinct successors. We prove that problem CDSDHP reduces to problem $F2|no-idle, no-wait|C_{\max}$. Correspondingly, we provide a new polynomially solvable special case of the Hamiltonian Path problem. In Section 3, we show that also the general $F|no-idle, no-wait|C_{\max}$ problem with m machines is polynomially solvable and present an $O(mn \log n)$ time solution approach. Finally, section 5 provides the unary NP-Hardness proof of problems $J2|no-idle, no-wait|C_{\max}$ and $O2|no-idle, no-wait|C_{\max}$.

2. Two-machine no-idle no-wait flow shop scheduling

In the $F2|no-idle, no-wait|C_{\max}$ problem, a set of n jobs is available at time zero. Each job J_j must be processed non-preemptively on two continuously available machines M_1, M_2 with known integer processing times $p_{1,j}, p_{2,j} > 0$, respectively. Each machine is subject to the so-called no-idle constraint, namely, it processes continuously one job at a time, and operations of each job cannot overlap. Each job is subject to the so-called no-wait constraint, namely, it cannot be idle between the completion of the first operation and the start of the second operation. All jobs are processed first on machine M_1 and next on machine M_2 and, given the no-wait constraint, the jobs sequences on the two machines must be identical. Let us denote by

$p(A) = \sum_{j=1}^{n} p_{1,j}$ the sum of processing times on the first machine and by $p(B) = \sum_{j=1}^{n} p_{2,j}$ the sum of processing times on the second machine. For any given sequence σ, $[j]_\sigma$ denotes the job in position j.

Consider Figure 1 which provides an illustrative example of a feasible no-idle, no-wait schedule for a 4-job problem.

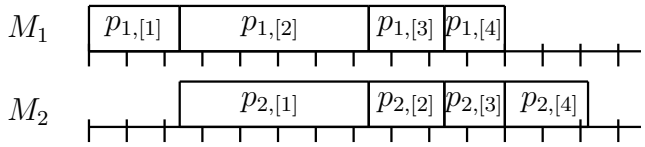


Figure 1: A no-idle no-wait schedule for a 2-machine flow shop

We point out that the $no-idle, no-wait$ requirement is very strong. Indeed, any feasible sequence σ for problem $F2|no-idle, no-wait|C_{\max}$, forces consecutive jobs to share common processing times in such a way that

$$p_{2,[j]_\sigma} = p_{1,[j+1]_\sigma} \quad \forall j \in \ ..., n-1. \tag{1}$$

As mentioned in the introduction, a related problem denoted by $F2|no-wait|\mathcal{G}$ is tackled in [15], where the aim is to minimize the number of interruptions (idle times) on M_2. We remark that this problem does not constitute a generalization of problem $F2|no-idle, no-wait|C_{\max}$ since an optimal solution with no interruptions of problem $F2|no-wait|\mathcal{G}$ may be non-optimal for problem $F2|no-idle, no-wait|C_{\max}$. Consider a $2-job$ instance with processing times $p_{1,1} = b$, $p_{2,1} = a$, $p_{1,2} = a$, $p_{2,2} = b$, with $b > a$. Then, sequence (J_1, J_2) is no-idle, no-wait, has makespan $C_{\max}^{(J_1,J_2)} = 2b + a$ and is optimal for problem $F2|no-wait|\mathcal{G}$ as it has no interruptions. However, it is not optimal for problem $F2|no-idle, no-wait|C_{\max}$ as sequence (J_2, J_1) is also no-idle, no-wait and has makespan $C_{\max}^{(J_2,J_1)} = 2a + b < 2b + a$.

2.1. The $F2|no-idle, no-wait|C_{\max}$ problem and the game of dominoes

We first provide a lemma on specific conditions of any instance of problem $F2|no-idle, no-wait|C_{\max}$ for which a feasible solution may exist.

Lemma 1.

(C1) A necessary condition to have a feasible solution for problem $F2|no-idle, no-wait|C_{\max}$ is that there always exists an indexing of the jobs so that $p_{1,2}, ...p_{1,n}$ and $p_{2,1}, ..., p_{2,n-1}$ constitute different permutations of the same vector of elements.

(C2) When the above condition (C1) holds, then

Case 1 if $p_{1,1} \neq p_{2,n}$, every feasible sequence must have a job with processing time $p_{1,1}$ in first position and a job with processing time $p_{2,n}$ in last position.

Case 2 if $p_{1,1} = p_{2,n}$ then there exists at least n feasible sequences each starting with a different job by simply rotating the starting sequence as in a cycle.

Proof. Condition (C1) trivially holds from expression (1). For condition C2, if $p_{1,1} \neq p_{2,n}$, then the processing time on the first machine of the job in first position must be equal to $p_{1,1}$ (similarly the processing time on the second machine of the job in last position must be equal to $p_{2,n}$) or else there is no way to fulfil expression (1). Besides, If $p_{1,1} = p_{2,n}$ and a feasible sequence σ exists fulfilling expression (1), then $p_{1,[1]_\sigma} = p_{2,[n]_\sigma}$ also holds. But then, any forward or backward rotation of σ also provides a feasible solution. □

Next, we show that for any feasible solution the makespan is determined by the processing time of the first job on the first machine plus $p(B)$. The following lemma holds.

Lemma 2. *The makespan of any feasible sequence σ is given by the processing time of the first job on the first machine plus the sum of jobs processing times on the second machine.*

Proof. As mentioned above, for any feasible sequence σ, we have $p_{2,[j]_\sigma} = p_{1,[j+1]_\sigma}\ \forall j \in 1, ..., n-1$. Correspondingly,

$$\begin{aligned} C_{[1]_\sigma} &= p_{1,[1]_\sigma} + p_{2,[1]_\sigma} \\ C_{[2]_\sigma} &= p_{1,[1]_\sigma} + p_{2,[1]_\sigma} + p_{2,[2]_\sigma} \\ &\dots \\ C_{[j]_\sigma} &= p_{1,[1]_\sigma} + \sum_{i=1}^{j} p_{2,[i]_\sigma} \end{aligned}$$

Hence,

$$C_{\max_\sigma} = C_{[n]_\sigma} = p_{1,[1]_\sigma} + \sum_{i=1}^{n} p_{2,[i]_\sigma} = p_{1,[1]_\sigma} + \sum_{i=1}^{n} p_{2,i}. \tag{2}$$

□

One may consider approaching problem $F2|no-idle, no-wait|C_{\max}$ by solving first problem $F2|no-idle, no-wait|\mathcal{G}$. Then, if a solution without

interruption is found, it is immediate to determine the optimal $F2|no-idle, no-wait|C_{\max}$ sequence by exploiting Lemmata 1, 2. Alternatively, $F2|no-idle, no-wait|C_{\max}$ has no feasible solution. This would induce, however, the same $O(n^2)$ complexity of the algorithm provided in [13] for problem $F2|no-idle, no-wait|\mathcal{G}$.

To reach a better complexity, we strongly exploit the specific no-idle/no-wait constraint that strictly links the $F2|no-idle, no-wait|C_{\max}$ problem to the game of dominoes. Dominoes are 1×2 rectangular tiles with each 1×1 square marked with spots indicating a number. A traditional set of dominoes consists of all 28 unordered pairs of numbers between 0 and 6. We refer here to the generalization of dominoes presented in [5] in which n tiles are present, each of the tiles can have any integer (or symbol) on each end and not necessarily all pairs of numbers are present.

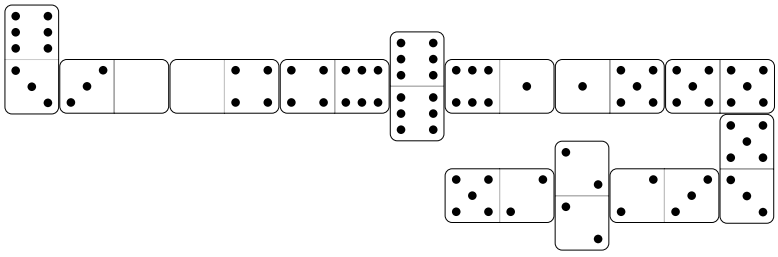


Figure 2: Solution of a Single Player Dominoes problem with 12 dominoes

In [5], it is shown that the Single Player Dominoes (SPD) problem (where a single player tries to lay down all dominoes in a chain with the numbers matching at each adjacency) is polynomially solvable as it can be seen as the solution of an Eulerian path problem on an undirected multigraph (in a way similar to the result provided in [13] but requiring lower complexity). Figure 2 shows the solution of an SPD problem with 12 tiles with numbers included between 0 and 6.

We refer here to the oriented version of SPD, denoted by $OSPD$, where all dominoes have an orientation, e.g. if the numbers are i and j, only the orientation $i \to j$ is allowed but not viceversa. The following Lemma holds.

Lemma 3. *Problem $OSPD$ is polynomially solvable.*

Proof. Following the proof for problem SPD in [5], we construct a multigraph G where vertices correspond to numbers and arcs (instead of edges) correspond to oriented dominoes. The rest works similarly. Hence, in this

case, the problem reduces to finding an Eulerian directed path in a directed multigraph which in turn is polynomially solvable [6]. □

The following proposition holds.

Proposition 1. *$F2|no-idle, no-wait|C_{\max}$ polynomially reduces to $OSPD$.*

Proof. By generating for each job J_j a related domino $\{p_{1,j}, p_{2,j}\}$, we know that any complete sequence of oriented dominoes in $OSPD$ corresponds to a feasible sequence for $F2|no-idle, no-wait|C_{\max}$ and viceversa. But then, due to Lemma 1, the jobs processing times either respect case 1 or case 2 of condition (C2). In case 1, the related sequence is optimal for $F2|no-idle, no-wait|C_{\max}$ as the processing time of the first job on the first machine is given and correspondingly, due to Lemma 2, the makespan is given. In case 2, we simply rotate the sequence in order to start with a job J_k having the smallest processing time on the first machine, that is job J_k with $p_{1,k} = \min_{i=1,..,n} p_{1,i}$. Correspondingly, the makespan $C_{\max} = p_{1,k} + \sum_{i=1}^{n} p_{2,i}$ is minimum and the related solution is optimal. □

By means of the reduction in Proposition 1, we propose an optimal algorithm, referred to as `AlgF2`, which solves problem $F2|no-idle, no-wait|C_{\max}$ in linear time.

Proposition 2. *Algorithm `AlgF2` solves problem $F2|no-idle, no-wait|C_{\max}$ in $O(n)$ time.*

Proof. Algorithm AlgF2 takes upon entry the set of n jobs and its associated multigraph G built as described in Lemma 3: each job processing time $p_{1,i}$ or $p_{2,i}$ corresponds to a vertex and there exists an arc from vertex k to vertex ℓ iff there exists a job J_i such that $p_{1,i} = k$ and $p_{2,i} = \ell$. Then, computing a feasible schedule to the $F2|no-idle, no-wait|C_{\max}$ problem reduces to computing, if it exists, a directed Eulerian path in G. The computation of an optimal solution is done by Algorithm AlgF2 directly by exploiting existence conditions of such a path.
Concerning the running time of the algorithm, it can be noticed that lines 1-5 can be executed in $O(n)$ time by an appropriate implementation. Lines 7 and 10 require to compute an Eulerian path in an oriented graph with n arcs, which can be done in $O(n)$ time ([6]). Notice that the construction of graph G, needed as an input to the algorithm, can also be built in $O(n)$ time. □

Consider the 9-job example of Table 1.
The corresponding optimal solution is provided in Figure 3.

Algorithm AlgF2: Solving the $F2|no-idle, no-wait|C_{\max}$ problem

Data: A set of n jobs to be scheduled with processing times $p_{1,i}$ and $p_{2,i}$, the associated multigraph G

Result: An optimal no-wait no-idle schedule, if it exists

$\mathcal{V} = \{p_{1,i}, p_{2,i}/i = 1, ..., n\}$;
$d^+(\alpha_i) = |\{j/p_{1,j} = \alpha_i\}|, \forall \alpha_i \in \mathcal{V}$;
$d^-(\alpha_i) = |\{j/p_{2,j} = \alpha_i\}|, \forall \alpha_i \in \mathcal{V}$;
$d(\alpha_i) = d^+(\alpha_i) - d^-(\alpha_i), \forall \alpha_i \in \mathcal{V}$;
$\mathcal{S} = \{\alpha_i/d(\alpha_i) \neq 0\}$;
if *(*$|\mathcal{S}|$*=0)* **then**
    Compute an Eulerian walk ew in G starting from vertex $\alpha_i = \min_{\alpha_k \in \mathcal{V}}(\alpha_k)$;
else
    if *(*$|\mathcal{S}| = 2$ *and (*$d(\alpha_1) = 1$ *and* $d(\alpha_2) = -1, \alpha_1, \alpha_2 \in \mathcal{S}$*))* **then**
        Compute an Eulerian walk ew in G from vertex α_1 to vertex α_2;
    else
        Exit: Problem infeasible;
    end
end
s^* is the optimal schedule, with $s^*[k]$ the job corresponding to arc $ew[k], \forall k = 1, ..., n$;
return s^*

i	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
$p_{1,i}$	5	3	4	6	1	5	3	2	4
$p_{2,i}$	3	4	6	1	5	3	2	4	5

Table 1: A 9-job instance of problem $F2|no-idle, no-wait|C_{\max}$

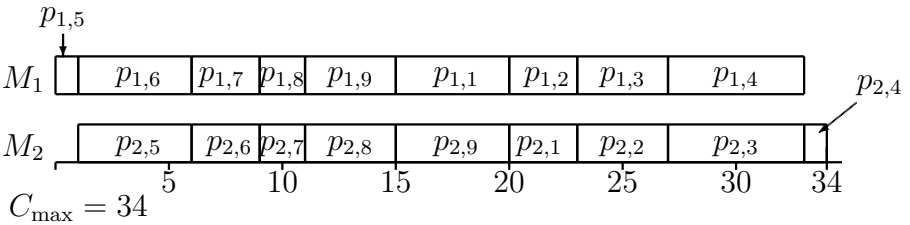


Figure 3: The optimal solution of the problem of Table 1

The input data and the solution of the $OSPD$ problem corresponding to the $F2|no-idle, no-wait|C_{\max}$ instance of Table 1 are provided in Figure 4.

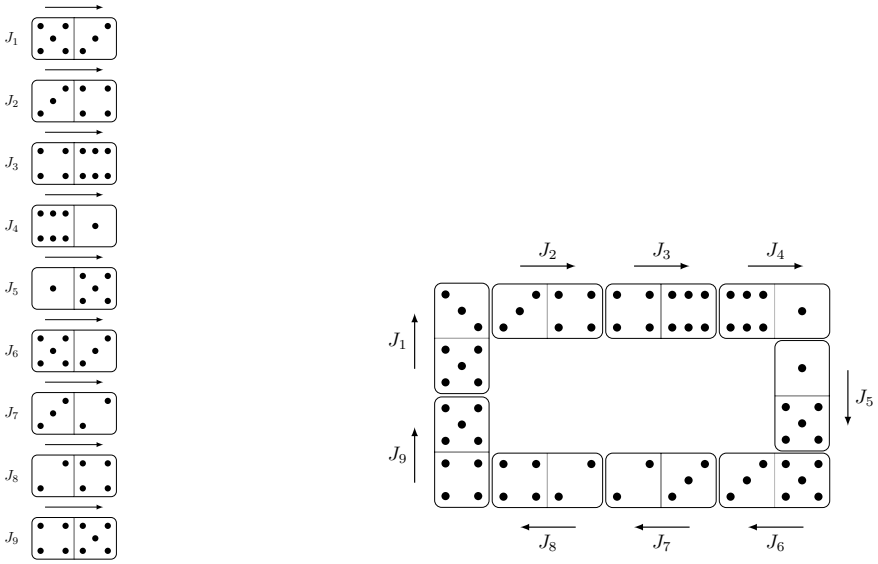


Figure 4: The dominoes corresponding to the flow shop instance of Table 1 and the related $OSPD$ problem solution

The multigraph computed by Algorithm AlgF2 on the flow shop instance of Table 1 is depicted in Figure 5.

2.2. The $F2|no-idle, no-wait|C_{\max}$ problem and the Common/Distinct Successors Directed Hamiltonian Path problem.

Problem $F2|no-idle, no-wait|C_{\max}$ is also linked to a special case of the Hamiltonian Path problem on a connected digraph. Consider a connected digraph $G(V, A)$ that has the following property: $\forall v_i, v_j \in V$, either $S_i \cap S_j = \emptyset$, or $S_i = S_j$, where S_i denotes the set of successors of vertex v_i. In

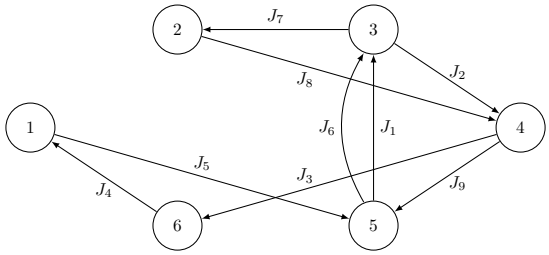


Figure 5: The multigraph originated by Algorithm AlgF2 on the flow shop instance of Table 1

other words, each pair of vertices either has no common successors or has all successors in common. Let indicate the Hamiltonian path problem in that graph as the Common/Distinct Successors Directed Hamiltonian Path (CDSDHP) problem. Notice that problem CDSDHP may have no feasible solution as indicated in Figure 6.

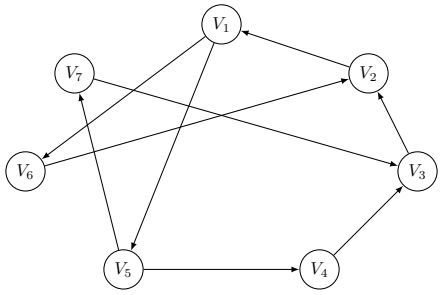


Figure 6: A non feasible instance of the CDSDHP problem

The following straightforward lemma holds.

Lemma 4. *Problem OSPD polynomially reduces to problem CDSDHP.*

Proof. Given an OSPD problem with n tiles, it is sufficient to generate a digraph $G(V, A)$ where each oriented tile corresponds to a vertex and there is an arc between two vertices if the corresponding tiles can match. Then, any complete sequence of oriented dominoes in OSPD corresponds to an hamiltonian path in the digraph and viceversa. □

From Property 1 and Lemma 4, we know that $F2|no-idle, no-wait|C_{\max}$ $\propto$ OSPD $\propto$ CDSDHP. Figure 7 depicts the CDSDHP instance corresponding to the OSPD problem of Figure 4 and to the flow shop instance of Table 1.

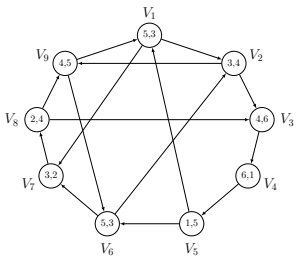


Figure 7: The CDSDHP instance corresponding to the OSPD problem of Figure 4 and to the flow shop instance of Table 1

The polynomial reduction between these problems actually works also in the opposite sense as indicated by the following Proposition 3.

For any given instance of problem CDSDHP, consider algorithm `AlgGenerF2` which constructs a related instance of problem $F2|no-idle, no-wait|C_{\max}$.

The following proposition holds.

Proposition 3. *CDSDHP polynomially reduces to problem* $F2|no-idle, no-wait|C_{\max}$.

Proof. For any given instance of CDSDHP, we generate an instance of $F2|no-idle, no-wait|C_{\max}$ according to Algorithm `AlgGenerF2`. Notice that the jobs processing times are generated in such a way that, if there is an arc from v_i to v_j, then, we have $p_{2,i} = p_{1,j}$. If a feasible sequence of $F2|no-idle, no-wait|C_{\max}$ exists, then, for each pair of consecutive jobs J_i, J_j with J_i preceding J_j, we have $p_{2,i} = p_{1,j}$ and, correspondingly, there is an arc from v_i to v_j. Thus, the corresponding sequence of vertices constitutes an Hamiltonian directed path for the considered instance of CDSDHP. Conversely, if an Hamiltonian directed path exists for the considered instance of CDSDHP, the corresponding sequence of jobs in problem $F2|no-idle, no-wait|C_{\max}$ is also feasible. □

Remark 1. *Problem CDSDHP is solvable in* $O(n^2)$ *time. Indeed, the* while*-loop in Algorithm AlgGenerF2 is applied at most* $O(n)$ *times and the predecessors and successors of any given* v_k *are at most* $O(n)$*. Correspondingly, Algorithm AlgGenerF2 has* $O(n^2)$ *complexity, while, from Proposition refpropAlgF2, we know that problem* $F2|no-idle, no-wait|C_{\max}$ *is solvable in linear time.*

3. *m*-machine no-idle no-wait flow shop scheduling

In this section we focus on the m-machine no-wait no-idle flowshop problem. Here each job J_j has processing times $p_{i,j}$, for all machines $M_1, ..., M_m$.

Algorithm AlgGenerF2: Generating an instance of problem $F2|no-idle, no-wait|C_{\max}$

Data: A digraph $G(V, A)$ such that $\forall v_i, v_j \in V$, either $S_i \cap S_j = \emptyset$, or $S_i = S_j$

Result: An instance of problem $F2|no-idle, no-wait|C_{\max}$ with n jobs where $n = |V|$

```
∀v_j ∈ V generate a corresponding job J_j;
count = 1;
while ∃v_k : p_{1,k} or p_{2,k} have not yet been determined do
    if (v_k has a self-loop) then
        p_{1,k} = p_{2,k} = count; count = count + 1;
    else
        if (both p_{1,k} and p_{2,k} have not yet been determined);
         then
            p_{1,k} = count; p_{2,k} = count + 1; count = count + 2;
        else
            if (p_{1,k} has not yet been determined);
             then
                p_{1,k} = count; count = count + 1;
            else
                if (p_{2,k} has not yet been determined);
                 then
                    p_{2,k} = count; count = count + 1;
                end
            end
        end
    end
    ∀v_j successor of v_k,  p_{1,j} = p_{2,k};
    ∀v_j predecessor of v_k,  p_{2,j} = p_{1,k};
    ∀v_j having common successors with v_k, p_{2,j} = p_{2,k};
    ∀v_j having common predecessors with v_k, p_{1,j} = p_{1,k};
end
return
```

Let us first introduce a generalized version of Lemma 1 illustrated in Figure 8.

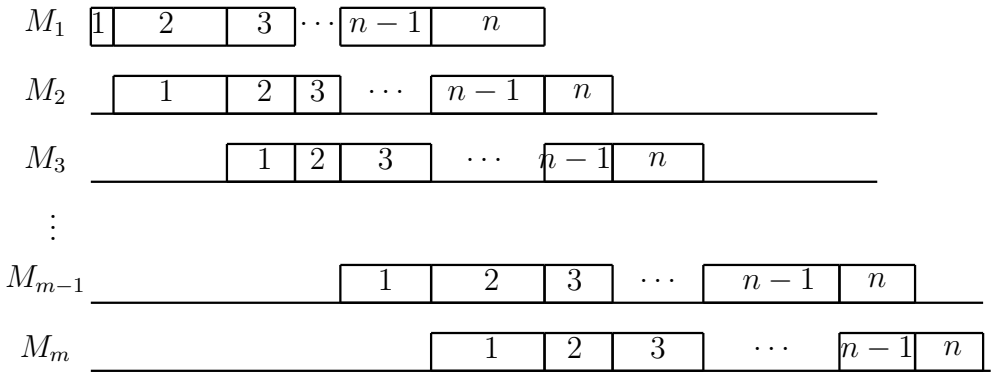


Figure 8: Solution of an m-machine flowshop problem

Lemma 5. *(C3) A necessary condition to have a feasible solution for problem* $F|no-idle, no-wait|C_{\max}$ *is that there always exists an indexing of the jobs so that,* $\forall j = 1, ..., m-1$, $p_{j+1,1}, ..., p_{j+1,n-1}$ *and* $p_{j,2}, ..., p_{j,n}$ *constitute different permutations of the same vector of elements.*

(C4) When the above condition (C3) holds, then

Case 1 if $(p_{1,1} \neq p_{2,n}$ *or* $p_{2,1} \neq p_{3,n}$ *or ... or* $p_{m-1,1} \neq p_{m,n})$*, every feasible sequence must have a job with processing times* $(p_{1,1}, ..., p_{m-1,1})$ *on machines* M_1 *to* M_{m-1} *in first position and a job with processing time* $(p_{2,n}, ..., p_{m,n})$ *on machines* M_2 *to* M_m *in last position.*

Case 2 if $(p_{1,1} = p_{2,n}$ *and* $p_{2,1} = p_{3,n}$ *and ... and* $p_{m-1,1} = p_{m,n})$ *then there exists at least n feasible sequences each starting with a different job by simply rotating the starting sequence as in a cycle.*

Proof. Similar to that of Lemma 1. □

From Lemma 5 and Figure 8 we can evince that in an optimal sequence if job J_i immediately precedes job J_k we must have $p_{j+1,i} = p_{j,k}, \forall j = 1, ..., m-1$. Consequently, for a feasible subsequence (J_ℓ, J_i, J_k) we must have:

$$[p_{2,\ell}; ...; p_{m,\ell}] = [p_{1,i}; ...; p_{m-1,i}] \text{ and } [p_{2,i}; ...; p_{m,i}] = [p_{1,k}; ...; p_{m-1,k}]. \quad (3)$$

This can be represented in terms of dominoes as indicated in Figure 9. The following proposition holds.

Proposition 4. *Problem* $F|no-idle, no-wait|C_{\max}$ *can be solved to optimality in* $O(mn \log n)$ *time.*

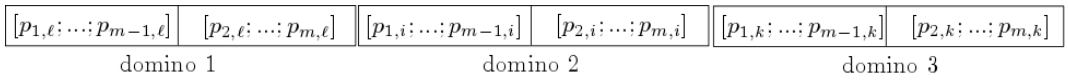


Figure 9: Three consecutive jobs / dominos

Proof. We show that any instance of the $F|no-idle, no-wait|C_{\max}$ problem can be reduced to another instance of problem $F2|no-idle, no-wait|C_{\max}$. The reduction works as follows. From any instance I_m of the m-machine problem, build $2n$ vectors $[p_{1,i};...;p_{m-1,i}]$ and $[p_{2,i};...;p_{m,i}]$, for each job $J_i \in I_m$. Sort these vectors according to the lexicographical increasing order of their values and let $v_{[t]}$ be the t-th vector in this order. This process requires $O((m-1)n\log n) \approx O(mn\log n)$ time to be done. Then, we create an instance I_2 of the 2-machine problem: $\forall J_i \in I_m$, we create a new job $J_\ell \in I_2$ such that $p_{1,\ell} = t_1$ and $p_{2,\ell} = t_2$ with $v_{[t_1]} = [p_{1,i};...;p_{m-1,i}]$ and $v_{[t_2]} = [p_{2,i};...;p_{m,i}]$. This process can be done in $O((m-1)n\log n) \approx O(mn\log n)$ time.

Finally, apply Algorithm AlgF2 to I_2. If instance I_2 is infeasible, then there does not exist a feasible solution to I_m due to equation 3 which cannot be answered for any triplet of consecutive jobs. If instance I_2 admits a feasible solution, then a feasible solution to I_m can be easily derived since jobs in I_2 correspond to jobs in I_m. Let s be the sequence of jobs of I_m obtained from the optimal solution returned by Algorithm AlgF2 on I_2. If Case 1 of (C4) holds for s then s is also optimal for I_m. Otherwise, it means that Case 2 holds and then the optimal solution s^* to I_m is obtained by considering the circular permutation of jobs in s where job J_ℓ with $\ell = argmin_{k=1...n}(\sum_{j=1}^{m-1} p_{j,k})$ is put first. The building of s^* can be done in $O((m-1)n) \approx O(mn)$ time. Consequently, the solution of the $F|no-idle, no-wait|C_{\max}$ problem can be done with an overall $O(mn\log n)$ time complexity.

□

4. Two-machine no-idle no-wait job shop scheduling

In this section we consider problem $J2|no-idle, no-wait|C_{\max}$. In the job shop configuration, differently from the flow shop, each job has its own processing routing. Also, when jobs with different processing routes are present, for every feasible solution, the jobs sequences on the two machines are necessarily different due to the no-wait constraint on the jobs. We establish the complexity of this problem by proving that it is NP-Hard in the strong sense. To this extent we consider the Numerical Matching with Tar-

get Sums (NMTS) problem that has been shown to be NP-complete in the strong sense [8].

NMTS
Instance: Disjoint sets X and Y, each containing m elements, a size $s(k) \in Z^+$ for each element $k \in X \cup Y$, and a target vector $< t_1, t_2, ..., t_m >$ with positive integer entries where $\sum_{i=1}^{m} t_i = \sum_{i=1}^{m} s(x_i) + \sum_{i=1}^{m} s(y_i)$.
Question: Can $X \cup Y$ be partitioned into m disjoint sets $D_1, D_2, ..., D_m$ each containing exactly one element from each of X and Y such that, for $1 \leq i \leq m$, $\sum_{k \in D_i} s(k) = t_i$?

Proposition 5. *Problem $J2|no-idle, no-wait|C_{\max}$ is NP-Hard in the strong sense.*

Proof. We show that NMTS polynomially reduces to problem $J2|no-idle, no-wait|C_{\max}$. For any given instance of NMTS, we generate in $O(m)$ time an instance of $J2|no-idle, no-wait|C_{\max}$ with $n = 3m$ jobs in the following way. Jobs $J_1, ..., J_m$ and jobs $J_{2m+1}, ..., J_{3m}$ follow the $M_1 \rightarrow M_2$ processing route. Jobs $J_{m+1}, ..., J_{2m}$ follow the $M_2 \rightarrow M_1$ processing route. Let us introduce $P = \sum_{i=1}^{m} t_i + \sum_{i=1}^{m} s(x_i) + \sum_{i=1}^{m} s(y_i)$. The processing times are as follows: $p_{1,1}, ..., p_{1,2m} = 1$; $p_{1,2m+i} = t_i + 3P, \ \forall i = 1, ..., m$; $p_{2,i} = s(x_i) + P, \ \forall i = 1, ..., m$; $p_{2,m+i} = s(y_i) + 2P, \ \forall i = 1, ..., m$; $p_{2,2m+1}, ..., p_{2,3m} = 2$.

From now on, we will denote jobs $J_1, ..., J_m$ as the x-jobs (as they relate to set X of NMTS). Similarly, jobs $J_{m+1}, ..., J_{2m}$ are denoted as the y-jobs and jobs $J_{2m+1}, ..., J_{3m}$ are denoted as the t-jobs. Also, we will refer to a pattern $\alpha - \beta - \gamma$ whenever a triplet $\alpha - \beta - \gamma$ is consecutively repeated on one machine. As an example, the triplet $x - t - y$ on M_1 indicates that a x-job is immediately followed by a t-job and then by a y-job on M_1. Correspondingly, the pattern $x - t - y$ on M_1 indicates that the sequence on M_1 is given by m consecutive triplets $x - t - y$.

The total load of machine M_1 is given by $L_1 = \sum_{i=1}^{3m} p_{1,i} = 2m + 3mP + \sum_{i=1}^{m} t_i$. Also, the total load of machine M_2 is given by $L_2 = \sum_{i=1}^{3m} p_{2,i} = 2m + 3mP + \sum_{i=1}^{m} s(x_i) + \sum_{i=1}^{m} s(y_i)$. Hence $L_1 = L_2$. Then, $L = \max\{L_1, L_2\} = L_1 = L_2$ constitutes a trivial lower bound on the makespan of the problem. We remark that, on the one hand, the processing times on M_1 are unitary for x-jobs and y-jobs and have value $> 3P$ for t-jobs. On the other hand, the processing times on M_2 have value 2 for t-jobs and value $> P$ for x-jobs and $> 2P$ for y-jobs respectively.

In any feasible no-idle no-wait schedule, on M_1 every t-job (except possibly the last one) must be immediately followed by a y-job first and then by

a x-job, that is a t-job necessarily induces on M_1 a triplet $t-y-x$. Indeed, two t-jobs cannot be adjacent on M_1, or else there would be idle time on M_2 between the completion of the first t-job and the start of the second t-job. Also, a t-job cannot be immediately followed by an x-job on M_1, or else the x-job would be waiting one unit of time between its completion on M_1 and its start on M_2. Finally, a triplet $t-y-y$ cannot hold as it would induce an idle time between the two y-jobs on M_1 and a triplet $t-y-t$ cannot hold as it would induce an idle time between the two t-jobs on M_2. As a consequence, as there are m t-jobs, a feasible schedule on M_1 follows either pattern $t-y-x$, or pattern $y-x-t$ or pattern $x-t-y$.

First consider pattern $y-x-t$ on M_1. This means that the first y-job, say job J_j, would be the first job in the sequence of M_1 and (due to the $M_2 \to M_1$ processing order) would be processed first on machine M_2 so that $C_{1,j} = C_{2,j} + 1$. Hence, the following job on M_2 is necessarily the x-job, say job J_i, with $C_{1,i} = C_{1,j} + 1 = C_{2,j} + 2$ which means that there is an idle time on M_2 between the end of job J_j and the beginning of job J_i. So, pattern $y-x-t$ on M_1 cannot lead to a feasible schedule.

Hence, every feasible sequence on M_1 either follows pattern $x-t-y$ or pattern $t-y-x$. In the first case, the pattern on M_2 is $x-y-t$ and the relevant Gantt diagram is shown in Figure 10 for the case of 6 jobs. In this case, the makespan is given by the load of M_2 plus the processing time on M_1 of the first x-job, that is $C_{\max} = L_2 + 1 = L + 1$. Further, note that each t-job J_k on M_1 starts and completes when an x-job J_j starts and a y-job J_i completes on M_2, respectively. Also, the sum $p_{2,i} + p_{2,j}$ of the processing times on M_2 of the x-job and the y-job is equal to the processing time $p_{1,k}$ on M_1 of the t-job. But then, it provides a true assignment to the corresponding NMTS problem, as $p_{2,i} + p_{2,j} = s(x_i) + s(y_j) + 3P$ and $p_{1,k} = t_k + 3P$, that is $s(x_i) + s(y_j) = t_k$.

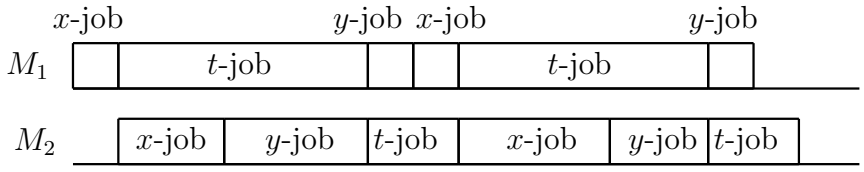


Figure 10: The $x-t-y$ on M_1 / $x-y-t$ on M_2 patterns sequence

In the latter case, the sequence on M_1 follows pattern $t-y-x$, that is starts with a t-job and ends with an x-job. The corresponding pattern on M_2 is $y-t-x$. Hence, as the x-jobs follow the $M_1 \to M_2$ processing route, the last

x-job on M_1 will then need to be processed also on M_2. Thus, if we denote by J_h this last x-job, the makespan will be $C_{\max} = L_1 + p_{2,h} > L_1 + P = L + P$ which is worse than the schedule obtained with pattern $x - t - y$ on M_1.

To conclude, if there exists a feasible schedule for the two-machine job shop problem with the shape $x - t - y$ on M_1 and $x - y - t$ on M_2 and value $= L + 1$, then this schedule is optimal and enables to derive in $O(m)$ time a yes answer to NMTS. Conversely, if there is no feasible solution to the job shop problem or the value is $> L + P$, then the answer to NMTS is no. This shows that the two-machine job shop problem is NP-hard.

□

5. Two-machine no-idle no-wait open shop scheduling

In this section we consider problem $O2|no-idle, no-wait|C_{\max}$. In the open shop configuration, no specific routing is assumed for each job which can be either started first on M_1 and then on M_2 or viceversa. We establish the complexity of this problem by proving that it is NP-Hard in the strong sense.

Proposition 6. *Problem $O2|no-idle, no-wait|C_{\max}$ is NP-Hard in the strong sense.*

Proof. We show that NMTS polynomially reduces to problem $O2|no-idle, no-wait|C_{\max}$ in a way similar to Proposition 5. From any given instance of NMTS, we generate in $O(m)$ time an instance of $O2|no-idle, no-wait|C_{\max}$ with $3m$ jobs having the same processing times as in the job shop case.

To show the reduction, we first show that no feasible solution exists with $C_{\max} = L$ and that, if a feasible solution exists with value $C_{\max} = L+1$, then a true assignment to the related NMTS problem holds and can be derived in $O(m)$ time from the solution of the $O2|no-idle, no-wait|C_{\max}$ problem.

In order to have $C_{\max} = L$, both machines should start processing at time 0. However, if an x-job (or a y-job) starts at time 0 on M_1, then no job can start at time 0 on M_2 or else the no-wait requirement on the x-job (the y-job) would be violated as all processing times on M_2 have length ≥ 2. Besides, if a t-job starts at time 0 on M_1, whatever job starting at time 0 on M_2 would then violate the no-wait requirement as any processing time on M_2 has length inferior to the processing time of any t-job on M_1. To fulfil the no-wait requirement, the considered job on M_2 would need to start at a

time $t_0 >> 0$ but then $C_{\max}$ would be much larger than L. Thus, no feasible solution exists with $C_{\max} = L$.

In order to have $C_{\max} = L + 1$, the first job cannot start on any machine after time 1. Notice, that the same argument explicited above rules out the possibility of having a t-job starting at time $t_0 \leq 1$ on M_1. Also, if an x-job (a y-job) starts at time 1 and completes at time 2 on M_1, in order to fulfil the no-wait requirement on the x-job (the y-job), a t-job should start at time 0 on M_2. But then, the t-job will start at time 2 on M_1 and the x-job (y-job) will start at time 2 on M_2. Correspondingly, to fulfil the no-idle no-wait requirement only a y-job (x-job) can be processed on M_2 once the x-job (y-job) is completed and the sum of the processing times of the x-job and y-job on M_2 must be equal to the processing time of the t-job on M_1. This corresponds to the start of a pattern $x - t - y$ ($y - t - x$) on M_1 and $t - x - y$ ($t - y - x$) on M_2 where all the t-jobs and y-jobs (x- jobs) follow the $M_2 \rightarrow M_1$ processing route and all the x-jobs (y-jobs) follow the $M_1 \rightarrow M_2$ processing route. The resulting makespan is $L + 1$. The corresponding two feasible schedules for a 6-job problem are depicted in Figure 11.

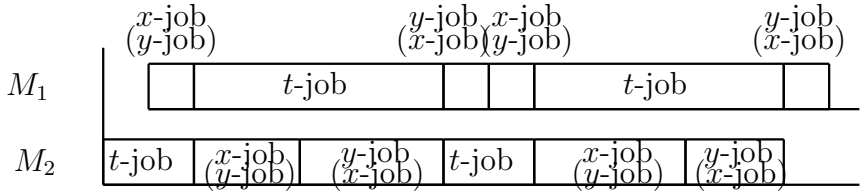


Figure 11: Two open shop patterns

Alternatively an x-job (y-job) must start at time 0 on M_1 and at time 1 on M_2 and, by applying the same reasoning it turns out that the only way to have a feasible schedule with $C_{\max} = L + 1$ is to stick to patterns $x - t - y$ ($y - t - x$) on M_1 and $x - y - t$ ($y - x - t$) on M_2. We remark that patterns $x - t - y$ on M_1 and $x - y - t$ on M_2 are the same patterns of the job shop problem, hence Figure 10 for a 6-job instance holds also here. Besides, by swapping every x-job with a y-job in Figure 10, we get the graphical representation of the solution with patterns $y - t - x$ on M_1 and $y - x - t$ on M_2.

By the same reasoning of the job shop case, we can see for all four cases, that if a feasible schedule with $C_{\max} = L + 1$ exists, then it provides also a true assignment to the corresponding NMTS problem by matching every

pair of x-job, y-job to the related t-job. Hence, given the solution of the generated instance of the $O2|no-idle, no-wait|C_{\max}$ problem, either a feasible solution exists with $C_{\max} = L+1$ and a true assignment to the corresponding NMTS problem is then obtained in $O(m)$ time, or else either $C_{\max} > L+1$ or no feasible solution exists for problem $O2|no-idle, no-wait|C_{\max}$ and correspondingly no true assignment exists for the NMTS problem. □

6. Conclusions

We considered no-idle/no-wait shop scheduling problems with makespan as performance measure. We firstly focused on the $F2|no{-}idle, no{-}wait|C_{\max}$ problem for which we established the connection to a special game of dominoes, namely the Oriented Single Player Dominoes, and provided an $O(n)$ algorithm to solve both problems to optimality. As a byproduct, we also considered a special case of the Hamiltonian Path problem (denoted as Common/Distinct Successors Directed Hamiltonian Path) and proved that it reduces to problem $F2|no{-}idle, no{-}wait|C_{\max}$. Correspondingly, we presented a new polynomially solvable special case of the Hamiltonian Path problem. Then, we extended our analysis to the more general $F2|no-idle, no-wait|C_{\max}$ problem showing that it can be transformed into a two-machine instance. Correspondingly, by applying the same approach as in the two-machine case, we showed that it can be solved with complexity $O(mn \log n)$. For this problem we have also proposed a vectorial version of the Oriented Single Player Dominoes problem with tiles composed by vectors of numbers. Finally, we proved that both the two-machine job shop and the two-machine open shop problems are NP-hard in the strong sense by reduction from the Numerical Matching with Target Sums.

Bibliography

[1] I. Adiri and D. Pohoryles. Flowshop / no-idle or no-wait scheduling to minimize the sum of completion times. *Naval Research Logistics*, 29, 495–504, 1982.

[2] A. Allahverdi. A survey of scheduling problems with no-wait in process. *European Journal of Operational Research*, 255, 665–686, 2016.

[3] P. Baptiste and K.H. Lee. A branch and bound algorithm for the $F|no-idle|C_{\max}$. *Proceedings of the International Conference on Industrial Engineering and Production Management*, 1, 429–438, 1997.

[4] P. Chrétienne. On scheduling with the non-idling constraint. *4OR*, 12, 101-121, 2014.

[5] E.D. Demaine, F. Ma and E. Waingarten. Playing Dominoes Is Hard, Except by Yourself. *FUN 2014, LNCS*, 8496, 137–146, 2014.

[6] H. Fleischner. Eulerian Graphs and Related Topics *Annals of Discrete Mathematics*, Part 1, Volume 2, Elsevier, 1991.

[7] M.R. Garey, D.S. Johnson and R. Sethi. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1, 117–129, 1976.

[8] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and CO., New York, 1982.

[9] K. Giaro. NP-hardness of compact scheduling in simplified open and flow shops. *European Journal of Operational Research* 130, 90–98, 2001.

[10] P.C. Gilmore, R.E. Gomory. Sequencing a one state variable machine: A solvable case of the traveling salesman problem. *Operations Research* 12, 655–679, 1964.

[11] Y. Goncharov and S. Sevastyanov The flow shop problem with no-idle constraints: A review and approximation. *European Journal of Operational Research* 196, 450–456, 2009.

[12] N.G. Hall and C. Sriskandarajah. A Survey of Machine Scheduling Problems with Blocking and No-Wait in Process. *Operations Research*, 44, 510–525, 1996.

[13] W. Höhn, T. Jacobs and N. Megow. On Eulerian extensions and their application to no-wait flowshop scheduling. *Journal of Scheduling*, 15, 295–309, 2012.

[14] S.M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1, 61–68, 1954.

[15] P.J. Kalczynski and J. Kamburowski. On no-wait and no-idle flow shops with makespan criterion. *European Journal of Operational Research*, 178, 677-685, 2007.

[16] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys. Sequencing and Scheduling: Algorithms and Complexity. *S.C. Graves, A.H.G. Rinnooy Kan and P. Zipkin (Eds.): Handbooks in Operations Research and Management Science vol 4: Logistics of Production and inventory, North-Holland, Amsterdam*, 445 – 522, 1993.

[17] S.S. Reddi and C.V. Ramamoorthy. On the flowshop sequencing problem with no-wait in process. *Operational Research Quarterly*, 23, 323–331, 1972.

[18] H. Röck. The three machine no-wait flowshop problem is NP-complete. *Journal of the Association for Computing Machinery*, 31, 336–345, 1984.