On Eulerian Extension Problems and their Application to Sequencing Problems

Wiebke Höhn[1⋆], Tobias Jacobs[2], and Nicole Megow[3]

[1] Technische Universität Berlin, Germany. hoehn@math.tu-berlin.de
[2] Albert-Ludwigs-Universität, Freiburg, Germany. jacobs@informatik.uni-freiburg.de
[3] Max Planck Institute for Informatics, Saarbrücken, Germany. nmegow@mpi-inf.mpg.de

Abstract. We introduce a new technique for solving several sequencing problems. We consider Gilmore and Gomory's variant of the Traveling Salesman Problem and two variants of no-wait flowshop scheduling, the classical makespan minimization problem and a new problem arising in the multistage production process in steel manufacturing.
Our technique is based on an intuitive interpretation of sequencing problems as Eulerian Extension Problems. This view reveals new structural insights and leads to elegant and simple algorithms and proofs for this ancient type of problems. As a major effect, we compute not only a single solution; instead, we represent the entire space of optimal solutions. For the new flowshop scheduling problem we give a full complexity classification for any machine configuration.

1 Introduction

We present a new technique for analyzing sequencing problems such as no-wait flowshop scheduling problems and a variant of the *Traveling Salesman Problem* (TSP). We show that these problems have a natural interpretation as *Eulerian Extension Problems*. This leads to new structural insights and new solution methods. On a high level view, for an instance of a sequencing problem we find a particular Eulerian graph in which all existing Eulerian circuits represent sequencing solutions with the same cost. In fact, we provide the entire set of optimal solutions, instead of just a single one. For a non-standard flowshop sequencing problem, which has not been investigated from an information theoretic point of view, we gain new structural insights which form the basis for fully settling the complexity status of any particular problem case.

A directed multi-graph $G = (V,E)$ is called *Eulerian* if it contains a cycle visiting each edge exactly once. A *Eulerian extension* is a set of additional edges E' for a given (not necessarily connected) multigraph $G = (V,E)$ such that $(V, E \cup E')$ is Eulerian. A *Eulerian Extension Problem* is, generally speaking, the problem of finding a Eulerian extension minimizing the total cost of additional edges E' according to some cost function. Notice that the classical *Chinese Postman Problem* (see e.g. [18]) is a special case in which the given graph is strongly connected and the cost function is based on the cost of paths in the graph. Eulerian extension problems are generally intractable

⋆ Supported by the German Research Foundation (DFG) as part of the priority programme "SPP 1307: Algorithm Engineering".

as straightforward reduction from TSP shows. We investigate Eulerian Extension Problems for special classes of cost functions arising in the context of the following classes of sequencing problems.

The TSP is one of the most intensively studied optimization problems; see e.g. [9]. In its full generality it is highly intractable, however special cases can be solved optimally in polynomial time. One of the most famous solvable subclasses was studied already four decades ago by Gilmore and Gomory [4]. In this problem variant, which we denote by G-TSP, each city i is associated with two numbers A_i and B_i for $i = 1,\ldots,n$. The cost for traveling from city i to city j is $\int_{B_i}^{A_j} f(x)dx$ if $A_j \geq B_i$ and $\int_{A_j}^{B_i} g(x)dx$ otherwise, where f,g are integrable functions satisfying $f(x)+g(x) \geq 0$, for any x.

Another classical sequencing problem is no-wait flowshop scheduling. In flowshop scheduling, we consider a production process where n jobs $J_1,\ldots,J_n$ must pass s production stages $L_1,\ldots,L_s$. Each job J_j consists of s operations each of which is dedicated to a specific stage L_i on which it must process for p_{ij} time units without preemption. Note that we consider operations with zero processing time as infinitely small operations which require a free machine. Each stage L_i has m_i identical parallel machines available. The jobs pass the production stages $L_1,L_2,\ldots,L_s$ in exactly this order. In a feasible no-wait flowshop schedule, there is no waiting time allowed between the execution of two consecutive operations of the same job. The goal is to minimize the makespan $C_{\max}$, that is the completion time of the last job. Following the classical three-field notation [5] we denote this problem by $\mathrm{F}\,|\,\mathrm{nwt}\,|\,C_{\max}$ if there is only a single processor available on each stage and by $\mathrm{FF}\,|\,\mathrm{nwt}\,|\,C_{\max}$ in the multiprocessor case. In case that the number of stages is fixed to s, we denote the corresponding processor environments by Fs and FFs, respectively.

In the single processor case, the no-wait condition implies the same job order on each machine stage. Thus, idle times are uniquely determined by a job order which leads to a natural interpretation as an asymmetric TSP. In fact, $\mathrm{F2}\,|\,\mathrm{nwt}\,|\,C_{\max}$ is well-known to be a special case of G-TSP [4, 11]: each job j can be interpreted as a city with $A_j = p_{1j}$ and $B_j = p_{2j}$, and the cost function is of Gilmore-Gomory type with $f \equiv 1$ and $g \equiv 0$.

A structurally quite different sequencing problem concerns no-wait flowshop scheduling with the objective of minimizing the number of *interruptions*, i.e., the number of continuous idle time intervals on the last stage L_s. Here, we refer to idle time as time intervals where a machine is not processing any job during the actual production process, i.e., the time before the first job and after the last job to be processed on the particular machine is not considered as idle time. We denote this new objective by $\mathscr{G}$.

This problem is motivated by a particular application in steel production, the continuous casting process, in which ladles of melted steel have to pass several production stages. The final stage, the casting machine, plays a special role: the steel must flow continuously into the casting machine. When the flow is broken (we call it *interruption*), then the casting machine must be stopped for maintenance and extensive cleaning. Therefore, practitioners call it their objective to minimize the number of interruptions.

Related work. Gilmore and Gomory [4] derived an algorithm to find an optimal solution for G-TSP. Its computation time $\mathscr{O}(n\log n)$ matches the lower bound on the worst case time complexity for any optimal algorithm solving this problem [13]. A slightly simplified variant of this algorithm has been presented by Vairaktarakis [16].

Due to its practical importance in production planning, most of the existing literature on no-wait flowshops addresses the objective of minimizing the makespan. For an extensive survey on various occurrences of no-wait constraints in production environments and previous theoretical work we refer to [6]. The special case of two-stage scheduling $\mathrm{F2}\,|\,\mathrm{nwt}\,|\,C_{\max}$ can be solved to optimality directly with Gilmore and Gomory's algorithm [4, 11]. The complexity status changes if there is more than one processor on one of the two stages; then the problem becomes strongly *NP*-hard [15].

The particular problem of scheduling the continuous casting process is investigated from a practical point of view e.g. in [7, 10, 14], where mathematical programming approaches as well as meta-heuristics and simulation are considered. To the best of our knowledge, there is no literature on theoretical investigations on the problem of minimizing the number of interruptions in a no-wait flowshop. The only related theoretical work we are aware of, enforces the aim for interruption-free scheduling as a hard constraint. In [2, 3], the authors give complexity and approximation results for openshop and flowshop problems with the objective to minimize the makespan when no interruption is allowed on *any* machine. This restriction is much stronger than what we aim for. In our application, idle times on other stages than the last one, the casting machine, do not incur extra cost. A more restricted variant of the same problem is considered in [17] with only two production stages and unit processing times on the first stage such that interruptions can occur only on the second stage. Even though this processor environment is close to our setting, we do not see how results could transfer between the makespan minimization problem and our problem of minimizing the number of interruptions.

Our contribution.[4] We interpret Gilmore and Gomory's TSP as a Eulerian Extension Problem with a specific cost function. We present an optimal algorithm which is much simpler than previously proposed ones [4, 16] and which admits a much simpler and more intuitive analysis (Section 2). Its worst case computation time is $\mathcal{O}(n \log n)$ which is best possible [13]. Moreover, our algorithm reveals a structural property that seems to be inherent in this kind of sequencing problems. Typically, an optimal solution is not unique. With our method we keep an implicit representation of the set of optimal solutions and defer the selection of a particular tour to the final part of the algorithm. This gives us the opportunity of conveniently accessing all optimal solutions. Besides the theoretical significance, this is meaningful to practical applications in which often a secondary optimization criteria plays a role. In this case, one may choose accordingly from the set of all optimal solutions regarding the first criteria.

Clearly, the optimality result for G-TSP applies directly to the classical two-stage flowshop scheduling problem $\mathrm{F2}\,|\,\mathrm{nwt}\,|\,C_{\max}$ since it is a special case of G-TSP. This is not the case for the new flowshop problem $\mathrm{F2}\,|\,\mathrm{nwt}\,|\,\mathscr{G}$. Nevertheless, we show an interpretation as a Eulerian Extension Problem with another appropriate cost function and derive an elegant and fast optimal algorithm (Section 3). In this case, we obtain an implicit representation of all optimal solutions in time $\mathcal{O}(n \log n)$, from which any particular optimum can be extracted in linear time. The main computation effort lies in sorting all processing times once, and thus, the algorithm runs in linear time if processing times are already sorted. Moreover, we solve optimally the generalized prob-

[4] Due to space limitation we cannot give full proofs of all results; missing ones can be found in the appendix.

lem FF2 | nwt | $\mathscr{G}$ with a single machine on the first stage, $m_1 = 1$. Notice, that this result is a sharp contrast to the makespan variant of the same problem which is known to be strongly *NP*-hard [15].

In Section 4 we show that the problem on three machine stages F3 | nwt | $\mathscr{G}$ is *NP*-hard. Again an interpretation as Eulerian Extension Problem is crucial. Finally, we complement our results with further findings that fully reveal the computational complexity of FFs | nwt | $\mathscr{G}$ for each value of s and each machine configuration $m_1, \ldots, m_s$.

2 Solving the Gilmore-Gomory Traveling Salesman Problem

We revisit the variant of the traveling salesman problem denoted by G-TSP in the previous section. Both algorithms, the original one by Gilmore and Gomory [4] and the improved method proposed by Vairaktarakis [16], are based on an interpretation of the problem as a bipartite matching problem, where each city i corresponds to an edge, the so-called *city edge*, from point A_i in the first partition to B_i in the second partition. The goal is to find a minimum cost matching E' such that the union of E' and the city edges constitute a cycle. Their algorithms first sort the vertices in both partitions, then compute a minimum cost matching and finally transform it into a cyclic one using an involved patching algorithm.

Similarly to these approaches, we also model a city i as an edge (A_i, B_i) and call it city edge, and we also seek to insert a minimum cost set E' of additional edges called *extension edges* such that the resulting graph contains a Eulerian cycle. The conceptual difference is that we do not consider the graph as bipartite, i.e. we allow E' to contain any possible edge.

Definition 1 (One-Dimensional Eulerian Extension Problem, 1DEE). *Given a finite directed graph $G = (V, E)$ where vertices in V are labeled with real numbers, and given integrable functions $f, g : \mathbb{R} \to \mathbb{R}$ with $f(x) + g(x) \geq 0$ for any $x \in \mathbb{R}$, the problem 1DEE denotes the task of finding a Eulerian Extension E' minimizing $\sum_{(u,v)\in E'} c(u,v)$, where*

$$c(u,v) = \begin{cases} \int_v^u f(x)dx & u \geq v \\ \int_u^v g(x)dx & v > u \ . \end{cases}$$

To avoid overloaded notation, we refer to a vertex by its label.

Theorem 1. *G-TSP is equivalent to 1DEE. The reductions can be done in linear time.*

Proof. Given an instance I of G-TSP, we construct the directed multigraph $G = (V, E)$ of a 1DEE instance I' as follows: let $V = \bigcup_{i=1}^{n} \{A_i, B_i\}$ and $E = \bigcup_{i=1}^{n} \{(A_i, B_i)\}$. A tour T for I can be directly translated into a Eulerian extension E' for I' having the same cost: For any city j that is visited immediately after city i, add an extension edge (B_i, A_j) to E'. The total cost of E' equals the cost of the tour T by definition of the cost functions.

Conversely, given a 1DEE instance I' with the graph $G = (V, E)$, we construct the G-TSP instance I by converting each edge $(u, v) \in E$ into a city j with $A_j = u, B_j = v$.

Consider a solution E' for I', and find an arbitrary Eulerian tour T' in $V = (G, E \cup E')$. We show that by interpreting the order of the city edges in T' as the order of the cities, gives a solution T to the G-TSP instance I of the same cost as E'.

Let $(w_1, w_2, \ldots, w_m)$ be a sub-path of T' traversing only extension edges. We assume w.l.o.g. that $w_1 < w_m$; the reverse case works analogously. Moreover, we can assume that $w_i < w_{i+1}$ for $i = 1, \ldots, m-1$. Suppose there exists values $w_i < w_{i+1}$ and $w_{i+1} > w_{i+2}$. Then simple calculations show that replacing edges $(w_i, w_{i+1}), (w_{i+1}, w_{i+2}) \in E'$ by (w_i, w_{i+2}) does not increase the cost of the Eulerian extension.

With this observation and the linearity of integrals, the total cost of extension edges $(w_1, w_2), (w_2, w_3), \ldots, (w_{m-1}, w_m)$ between successive city edges $(*, w_1)$ and $(w_m, *)$ equals $c(w_1, w_m)$. This implies by definition of the cost function that the costs of solutions T and T' are equal. □

Motivated by the cost function of 1DEE that is defined on vertex labels, we introduce the concept of *minimal edges*. Consider a linear ordering of vertices in non-decreasing order of labels. Then we call an edge (u, v) *minimal*, if u and v are direct neighbors in this ordering.

Lemma 1. *Given a Eulerian extension E' for an instance of 1DEE, there is a Eulerian extension E'' satisfying the following properties: E'' contains only minimal extension edges, E' and E'' have the same cost, and for each Eulerian cycle in $E \cup E'$ there is one in $E \cup E''$ where the city edges occur in the same order.*

Proof. Let (u, v) be a non-minimal edge in E', i.e., there is a vertex w with $u < w < v$ or $u > w > v$. We replace (u, v) by (u, w) and (w, v) to obtain E''. This preserves the balance of indegree and outdegree for each vertex, and due to the linearity of the integral the overall cost does not change. Repeatedly performing this kind of operation for each non-minimal edge we obtain a Eulerian extension that contains only minimal edges. Analogously, we can replace each edge of a Eulerian cycle in $E \cup E'$ by the corresponding path of minimal edges in $E \cup E''$. This preserves the order of city edges in the cycle. □

Let $\mathrm{indeg}(v)$ and $\mathrm{outdeg}(v)$, $v \in V$, denote the indegree and outdegree of v, respectively. The following sufficient condition for a Eulerian graph is well-known; see [18].

Lemma 2. *A graph $G = (V, E)$ is Eulerian if and only if it is connected and* $\mathrm{indeg}(v) = \mathrm{outdeg}(v)$*, for all $v \in V$.*

Let $\mathrm{indeg}(V')$ denote the indegree of a subset $V' \subseteq V$, i.e., the number of edges $(u, v) \in E$ with $u \notin V'$ and $v \in V'$. Let the outdegree $\mathrm{outdeg}(V')$ be defined analogously. We state a necessary condition based on indegree and outdegree of vertex subsets.

Lemma 3. *If a graph $G = (V, E)$ is Eulerian, then* $\mathrm{indeg}(V') = \mathrm{outdeg}(V')$ *for any subset of vertices $V' \subseteq V$.*

The basic idea for our algorithm solving 1DEE is as follows: We restrict ourselves to solutions consisting only of minimal edges. First, we identify an edge set that *any* such feasible solution must contain. We obtain a set of connected components each of

which is Eulerian. In the second step, we add a minimum cost edge set that connects all components while keeping them Eulerian.

Algorithm 1

1: Sort all vertices in V in non-increasing order of their labels.
2: Let $E' = \emptyset$. For $i := 1$ to $n-1$ do
   Consider v, the i-th vertex in the ordering, and denote its direct successor by v'. Compute $b(v) = \text{indeg}(v) - \text{outdeg}(v)$ with respect to the graph $(V, E \cup E')$.
   If $b(v) > 0$, then add $b(v)$ copies of the minimal edge (v, v') to E', otherwise, add $-b(v)$ copies of the minimal edge (v', v) to E'.
3: Let $G_1, \ldots, G_k$ denote the connected components of the graph $(V, E \cup E')$. If $k \geq 2$, construct the undirected, connected component graph H by contracting each G_i, $i = 1, \ldots, k$, to a single vertex in H. For any minimal edge $(u, v) \notin E'$ with $u \in G_i$ and $v \in G_j$ we add an undirected edge (G_i, G_j) to H with weight $c(u, v) + c(v, u)$.
   Compute a Minimum Spanning Tree (MST) T in H. For each edge $(u, v) \in T$, add both associated directed, minimal edges (u, v) and (v, u) to E'.

Lemma 4. *Algorithm 1 chooses in Step 2 only edges that are necessary for any feasible Eulerian extensions that consist only of minimal edges.*

Proof. We prove that after any iteration of Step 2 in Algorithm 1, E' consists only of necessary edges. In any iteration, the algorithm inserts $|b(v)|$ edges between a vertex v and its direct successor v' in the given ordering. Consider some iteration and the corresponding vertices v and v'. Let $V' = \{u \in V \mid u \leq v\}$. Lemma 3 provides the necessary condition that $\text{indeg}(V') = \text{outdeg}(V')$ in the given graph when enhanced by any Eulerian extension. Before inserting additional edges, we have $\text{indeg}(u) = \text{outdeg}(u)$ for any $u \in V' \setminus \{v\}$ in the current graph $(V, E \cup E')$. Since we are restricted to minimal extension edges, any Eulerian extension must add $|\text{indeg}(V') - \text{outdeg}(V')|$ appropriately oriented edges between v and v'. The algorithm inserts exactly the required number of edges since $|\text{indeg}(V') - \text{outdeg}(V')| = |\text{indeg}(v) - \text{outdeg}(v)| = b(v)$. □

Theorem 2. *Algorithm 1 solves 1DEE optimally in time $\mathcal{O}(n \log n)$.*

Proof. By Lemma 1 we can restrict our attention to Eulerian extensions consisting only of minimal edges. Let E_1 denote the set of extension edges chosen by Algorithm 1 by the end of Step 2. Each vertex $i = 1, \ldots, n-1$ has equal in- and outdegree in the graph $(V, E \cup E_1)$. By the *handshaking argument* this also holds for the last vertex. If $(V, E \cup E_1)$ is connected, then it is Eulerian by Lemma 2. Applying additionally Lemma 4 we have proven that E_1 is a necessary and sufficient set of edges and thus optimal.

Otherwise, $(V, E \cup E_1)$ consists of multiple strongly connected components $G_1, \ldots, G_k$ each of them being Eulerian. By Lemma 4 any optimal solution must contain E_1. The algorithm now finds a set of additional minimal extension edges of minimum total cost that connects all components and ensures that the graph is Eulerian. Notice, that if we add a single extension edge (u, v) to connect two components G_i and G_j, then by Lemma 3 and the restriction to minimal edges (Lemma 1), we additionally need to add the reverse edge (v, u) to ensure that the resulting graph is Eulerian. Therefore our algorithm considers in Step 3 all relevant edges to connect $G_1, \ldots, G_k$ and

assigns cost for adding both, forward and backward edge. Thus, the MST solution on the accordingly constructed graph H corresponds to a minimum cost edge subset that connects all components and keeps the graph Eulerian.

Concerning the runtime of the algorithm, we note that the connected component graph H in Step 3 can be constructed in $\mathcal{O}(n)$ because the vertices are already sorted by their labels (Step 1). Consider all vertices in the given order starting from the vertex with smallest index. Suppose $v \in G_i$. If v's direct successor v' belongs to a different connected component $G_j \neq G_i$, then add an undirected edge between the corresponding vertices G_i and G_j in H. With this observation, the runtime of our algorithm is dominated by sorting the input and computing an MST which can be done in $\mathcal{O}(n \log n)$. □

The reduction from G-TSP to 1DEE (Proof of Theorem 1) implies that for each solution to G-TSP there is a Eulerian tour in a solution to the corresponding 1DEE instance. In this Eulerian tour, the city edges are traversed in exactly the same order as the cities are traversed in the G-TSP solution. By Lemma 1 there is a Eulerian extension consisting only of minimal edges which admits such a tour. This implies that we do not loose any optimal G-TSP tour when restricting ourselves to minimal edge extensions.

If the minimum spanning tree computed in Step 3 of Algorithm 1 is unique, then the optimal set of Eulerian extension edges E' is unique under the restriction to minimal edges. Since this does not restrict the space of optimal solutions for the corresponding G-TSP instance, each Eulerian cycle in the extended graph $(V, E \cup E')$ corresponds to an optimal solution for G-TSP. If there is more than one MST, then each of them represents a subset of all optimal G-TSP solutions.

3 Solving Two-Stage No-wait Flowshop Problems

While the makespan minimization problem $\mathrm{F2}\,|\,\mathrm{nwt}\,|\,C_{\max}$ can be solved directly as a special case of G-TSP with Gilmore-Gomory type cost functions $f \equiv 1$, $g \equiv 0$, there is no way to express the interruption related objective function $\mathcal{G}$ as special functions f and g in G-TSP. Nevertheless, we show that the problem $\mathrm{F2}\,|\,\mathrm{nwt}\,|\,\mathcal{G}$ has an interpretation as a Eulerian Extension Problem, which leads to a fast and elegant algorithm.

We define a cost function for a Eulerian Extension Problem in which extension edges (u,v) with $u < v$ account for interruptions on the second machine. We call such extension edges *up edges*. We denote extension edges (u,v) with $u > v$ as *down edges*.

Definition 2 ($\mathcal{G}$-related One-Dimensional Eulerian Extension Problem, $\mathcal{G}$-1DEE). *Given a finite directed graph $G = (V,E)$ where the vertices in V are labeled with real numbers, the problem $\mathcal{G}$-1DEE is to find a Eulerian extension for G minimizing the number of up edges.*

Theorem 3. $\mathrm{F2}\,|\,\mathrm{nwt}\,|\,\mathcal{G}$ *is equivalent to $\mathcal{G}$-1DEE. The reductions can be done in linear time.*

Proof sketch. An instance $I' = (V,E)$ of $\mathcal{G}$-1DEE is constructed from an $\mathrm{F2}\,|\,\mathrm{nwt}\,|\,\mathcal{G}$ instance I by defining $V = \bigcup_{i=1}^{n} \{p_{1i}, p_{2i}\}$ and by adding a so-called *job edge* (p_{1i}, p_{2i})

to E for each job J_i, $i = 1, \ldots, n$. Consider an optimal solution E' to I'. The set E' is converted into a solution to I by scheduling the jobs in the order in which the corresponding job edges appear in some Eulerian cycle in $(V, E \cup E')$. An interruption in the schedule occurs whenever an up edge is traversed between two job edges. The schedule is constructed such that the first job corresponds to a job edge traversed after some up edge in the cycle (if $k \neq 0$). Thus, this specific up edge does not account for an interruption and we obtain a schedule causing $\max\{k-1, 0\}$ interruptions. The schedule is optimal, because we can show that any schedule can be transformed back into a Eulerian tour in the same way. □

As in Section 2, we call a down edge (u, v) *minimal* if u and v are direct neighbors in a linear ordering of V by non-decreasing labels. Furthermore, we denote an up edge (u, v) as *maximal* if u has the minimum label and v has the maximum label in V.

Lemma 5. *Given a Eulerian extension E' for an instance of $\mathcal{G}$-1DEE with $G = (V, E)$, there is a Eulerian extension E'' satisfying the following properties: E'' contains only minimal down edges and maximal up edges, E' and E'' have the same cost, and for each Eulerian cycle in $(V, E \cup E')$ there is one in $(V, E \cup E'')$ where the edges from E appear in the same order.*

Algorithm 2

1: Sort all vertices in V in non-decreasing order of their labels.
2: Let v_i be the i-th vertex in the ordering and let $V_i := \{u \in V \mid u \leq v_i\}$. Compute $b_{\max} := \max_{i=1,\ldots,n-1}\{0, b(v_i)\}$, where $b(v_i) := \text{indeg}(V_i) - \text{outdeg}(V_i)$.
3: Initialize E' as the set containing $b_{\max}$ copies of the maximal up edge (v_1, v_n).
4: For $i := 1$ to $n-1$, add $b_{\max} - b(v_i)$ minimal down edges (v_{i+1}, v_i) to E'.
5: If $(V, E \cup E')$ is not strongly connected, add one further maximal up edge (v_1, v_n) and minimal down edges $\{(v_{i+1}, v_i) \mid 1 \leq i \leq n-1\}$ to E'.

With similar techniques as in the previous section we prove the following result.

Lemma 6. *Algorithm 2 chooses in Step 3 and 4 only edges that are necessary for any feasible Eulerian extension that consist only of minimal down edges and maximal up edges. The two steps effectuate that* $\text{indeg}(v) = \text{outdeg}(v)$ *for each $v \in V$ in $(V, E \cup E')$.*

Proof. Let v_i be the vertex maximizing $b(v_i)$ in Step 2 of the algorithm. Lemma 3 states that there must be $\max\{0, b(v_i)\} = b_{\max}$ edges from V_i to $V \setminus V_i$ in any feasible Eulerian extension. As these edges are up edges, they have to be maximal due to our restriction. They are inserted by the algorithm in Step 3.

After Step 3, we have $\text{indeg}(V_i) - \text{outdeg}(V_i) = b(v_i) - b_{\max} \leq 0$ in the graph $(V, E \cup E')$ for each $i = 1, \ldots, n-1$. So from Lemma 3 follows that for the graph to be Eulerian there must be $b(v_i) - b_{\max}$ additional edges from $V \setminus V_i$ to V_i. Such edges are down edges, and due to our restriction to minimal ones, they must be copies of the edge (v_{i+1}, v_i). Inserting them is exactly what Algorithm 2 does in Step 4.

As soon as Step 4 has been executed, we have $\text{indeg}(V_i) = \text{outdeg}(V_i)$ for $1 \leq i \leq n$, and in particular $\text{indeg}(v_1) = \text{outdeg}(v_1)$. Since $\text{indeg}(V_i) - \text{outdeg}(V_i) = \sum_{j=1}^{i} \text{indeg}(v_j) - \text{outdeg}(v_j)$ (see proof of Lemma 3), it follows inductively that $\text{indeg}(v_i) = \text{outdeg}(v_i)$ for all i. □

Theorem 4. *Algorithm 2 solves $\mathscr{G}$-1DEE optimally in time $\mathcal{O}(n \log n)$. Steps 2-5 take only linear time.*

Proof. By Lemma 5 it suffices to consider only maximal up edges and minimal down edges as extension edges. Let E_1 be the set of edges chosen by the algorithm by the end of Step 4. By Lemma 6, E_1 is a subset of any feasible Eulerian Extension for G, and we know that the indegree and outdegree of each node is balanced in $(V, E \cup E_1)$. If this graph is strongly connected, it is Eulerian (Lemma 2), and thus, Algorithm 2 is optimal.

Otherwise, at least one additional extension edge, i.e., either a maximal up edge or a minimal down edge, must be added. Suppose, we add a minimal down edge, say (v_{i+1}, v_i), then we have $\text{indeg}(V_i) - \text{outdeg}(V_i) = 1$ in the resulting graph. From Lemma 3 follows that we need an additional up edge for re-establishing balanced indegree and outdegree for each node. Thus, to establish connectivity in $(V, E \cup E_1)$ it is necessary to add an up edge. It is easy to see that with one additional up edge the set of minimal down edges inserted in Step 5 of the algorithm is necessary and sufficient to re-establish the balancedness of each node's indegree and outdegree. The resulting graph $(V, E \cup E')$ contains the cycle $v_1, v_n, v_{n-1}, \ldots, v_1$ and is therefore strongly connected.

The runtime of the algorithm is dominated by the time for sorting, $\mathcal{O}(n \log n)$, in Step 1. The remaining steps require linear computation effort; in particular, values $b(v_i)$ can be computed in Step 2 as $b(v_1) = \text{indeg}(v_1) - \text{outdeg}(v_1)$ and $b(v_i) = b(v_{i-1}) + \text{indeg}(v_i) - \text{outdeg}(v_i)$ in linear time. □

We remark that the optimal solution to $\mathscr{G}$-1DEE is always unique, because the objective function only depends on the number of up edges, and – assuming that all up edges are maximal – the set of down edges is uniquely determined by their number. As a consequence of the reduction given in the proof of Theorem 3 and Lemma 5, the extension E' computed by Algorithm 2 implicitly represents all optimal schedules.

The above algorithm can be applied also to the two-stage flowshop problem to minimize the number of interruptions with more than one machine on the second stage.

Theorem 5. *Any problem instance I of* $\text{FF2} \,|\, \text{nwt} \,|\, \mathscr{G}$ *with a single processor on the first stage, $m_1 = 1$, can be solved optimally in time $\mathcal{O}(n \log n)$.*

Proof sketch. Instances I of $\text{FF2} \,|\, \text{nwt} \,|\, \mathscr{G}$ with $m_2 > 1$ can be solved by considering the same set of jobs as an instance I' of $\text{F2} \,|\, \text{nwt} \,|\, \mathscr{G}$. Exactly $m_2 - 1$ interruptions caused by an optimal schedule S' for I' can be avoided by migrating to another machine on the second stage, so an optimal schedule S for I has exactly $m_2 - 1$ interruptions less than S'. □

4 Complexity of Minimizing the Number of Interruptions

In contrast to the polynomial time solvable problems with two machine stages in Section 3, the problem becomes strongly *NP*-hard in any other cases.

Theorem 6. *The problem* $\text{FF}s \,|\, \text{nwt} \,|\, \mathscr{G}$ *is strongly NP-hard for any constant number of stages $s \geq 3$ and arbitrary constant numbers of machines. The same is true for* $\text{FF2} \,|\, \text{nwt} \,|\, \mathscr{G}$ *with $m_1 > 1$.*

The proof follows by combining *NP*-hardness results for four particular problem classes (machine configurations). The problem $\mathrm{F3}\,|\,\mathrm{nwt}\,|\,\mathscr{G}$ plays the main role with respect to the focus on Eulerian Extensions of this work. Therefore, we consider here only this case while the remaining result can be found in the Appendix.

To show the complexity result, we consider a natural two-dimensional variant of $\mathscr{G}$-1DEE, in which vertices have two labels which can be seen as points in $\mathbb{R}^2$.

Definition 3 (Two-Dimensional Eulerian Extension(2DEE)). *Given a directed graph $G = (V,E)$ with vertices $V \subset \mathbb{R}_0^+ \times \mathbb{R}_0^+$, determine whether there exists a Eulerian extension E' for G using only* down edges $\{(u,v) \mid u,v \in V, u \geq v \text{ component-wise}\}$.

We provide a reduction from the Three-Dimensional Matching Problem (see e.g. [1]) to show the following result.

Theorem 7. *The problem 2DEE is strongly NP-complete.*

Now we are ready to show NP-completeness for $\mathrm{F3}\,|\,\mathrm{nwt}\,|\,\mathscr{G}$. We give a reduction to the decision variant in which we ask if a interruption-free solution exists. We denote the decision problem by $\mathscr{E}_0\,(\mathrm{F3}\,|\,\mathrm{nwt}\,|\,\mathscr{G}\,)$. Note, that *NP*-completeness of such a problem implies obviously *NP*-hardness and also inapproximability of the corresponding optimization problem. However, we can show that the optimization problem remains strongly *NP*-hard under the assumption that every solution has at least one interruption.

Theorem 8. *It is NP-complete to decide whether there is an interruption-free schedule for an instance of* $\mathrm{F3}\,|\,\mathrm{nwt}\,|\,\mathscr{G}$.

We show the *NP*-completeness of the decision problem $\mathscr{E}_0\,(\mathrm{F3}\,|\,\mathrm{nwt}\,|\,\mathscr{G}\,)$ by reduction from 2DEE. Unlike in the one-dimensional case, there is no one-to-one correspondence between instances of 2DEE and $\mathscr{E}_0\,(\mathrm{F3}\,|\,\mathrm{nwt}\,|\,\mathscr{G}\,)$. Therefore, our proof has the following structure: We first give an interpretation of our scheduling problem as an extension problem. Then we fix a set of properties that are only satisfied by 2DEE instances representing a scheduling problem instance in that way. Finally, we prove that any 2DEE instance can be transformed into an equivalent instance satisfying these properties.

Proof (of Theorem 8). Consider a set of jobs $J_1,\ldots,J_n$. A schedule without interruptions corresponds to a permutation $J_{\sigma(1)},\ldots,J_{\sigma(n)}$ of the jobs, where for $1 \leq i < n$ job $J_{\sigma(i+1)}$ can be scheduled after $J_{\sigma(i)}$ without causing an idle time on the third machine. This is the case if and only if $p_{3\sigma(i)} \geq p_{2\sigma(i+1)}$ and $p_{3\sigma(i)} + p_{2\sigma(i)} \geq p_{2\sigma(i+1)} + p_{1\sigma(i+1)}$.

In terms of our extension problem, this means that the point $(p_{2\sigma(i+1)}, p_{2\sigma(i+1)} + p_{1\sigma(i+1)})$ is reachable from the point $(p_{3\sigma(i)}, p_{3\sigma(i)} + p_{2\sigma(i)})$. We associate every job J_j with an edge from $(p_{2j}, p_{2j} + p_{1j})$ to $(p_{3j}, p_{3j} + p_{2j})$. In the consequence, there is an interruption-free schedule of $J_1,\ldots,J_n$ if and only if the induced graph (V_0,E_0) admits an extension E' such that there is a path traversing each edge exactly once. This is the case if and only if there is a Eulerian extension of the graph $(V,E) = (V_0 \cup \{v_{\min}, v_{\max}\}, E_0 \cup \{(v_{\min}, v_{\max})\})$, where $v_{\min}$ ($v_{\max}$) is such that it is smaller (greater) than all other elements of V in both coordinates.

We call a graph $G^* = (V^*,E^*)$ with $V^* \subset \mathbb{R}_0^+ \times \mathbb{R}_0^+$ *legal*, if it represents a scheduling instance in the way we have just described. Respectively, an edge is called legal if it

represents some job from a scheduling instance. It is not hard to observe that for an edge to be legal it suffices that it has the form $((x,y),(x',x+x'))$ with $x \leq y$. In other words, the source and sink vertex of a legal edge are above the bisectrix, and for a given source there is only one degree of freedom for the choice of the sink. For a graph to be legal it suffices that it is induced by $\hat{E} \cup \{(v_{\min}, v_{\max})\}$, where $\hat{E}$ is a set of legal edges, and $v_{\min}$ and $v_{\max}$ are like defined in the preceding paragraph.

We complete our reduction by describing in Lemma 7 how to *legalize* an arbitrary instance $G = (V,E)$ of 2DEE, that is, to transform it into a legal instance $G^* = (V^*,E^*)$, where G admits a Eulerian extension if and only if G^* does. □

Lemma 7. *Each instance of 2DEE can be legalized in polynomial time.*

Proof. First, we ensure that every vertex is above the bisectrix and no vertex has coordinates $(0,0)$. This is achieved by vertically shifting the whole graph by $x_{\max} = \max\{x \mid (x,y) \in V\} + 1$. Technically, we obtain the graph $G_1 = (V_1,E_1)$ by adding $x_{\max}$ to the second coordinate of every vertex. As this does not change the reachability relation between any pair of vertices, G and G_1 are equivalent in the sense of 2DEE.

In the second transformation step, we eliminate all illegal edges. Let edge $(u,v) = ((u_x,u_y),(v_x,v_y)) \in E_1$ be illegal. Let $y_{\max} = \max\{y \mid (x,y) \in V_1\} + 1$. We enhance V_1 by the vertices $w_1 = (y_{\max} - u_x, y_{\max})$, $w_2 = (0, y_{\max})$, $w_3 = (y_{\max}, y_{\max})$ and $w_4 = (v_y - v_x, y_{\max})$. Then, we replace (u,v) with the edges (u,w_1),(w_2,w_3) and (w_4,v), see Fig. 2.

Straightforward observation shows that all new edges are legal and any possible tour in a Eulerian extension must traverse the path u,w_1,w_2,w_3,w_4,v. Thus, the modified graph admits a Eulerian extension if and only if G_1 does.

We obtain the graph $G_2 = (V_2,E_2)$ by iteratively eliminating every illegal edge from G_1. Note that each such elimination causes $y_{\max}$ to increase by one. Still G_2 does not necessarily represent a flowshop scheduling instance, as there has to be an edge $(v_{\min}, v_{\max})$. We take care of this requirement in the last step of transformation. Let $u = (u_x,u_y) \in V_2$ be an arbitrary vertex. We insert $v_{\min} = (0,0)$ and $v_{\max} = (y_{\max}, y_{\max})$ into V_2, where $y_{\max} = \max\{y \mid (x,y) \in V_2\} + 1$. Furthermore, we insert w_1, w_2 and w_4 defined like in the transformation from G_1 to G_2. Note that w_3 has already been inserted as $v_{\max}$. Then, the edges (u,w_1), $(w_2,v_{\min})$, $(v_{\min},v_{\max})$ and (w_4,u) are added to E_2.

As the new vertices are above the bisectrix, the new edges are legal, and the resulting graph $G^* = (V^*,E^*)$ contains an edge from $v_{\min}$ to $v_{\max}$, it represents an instance of $\mathrm{F3}\,|\,\mathrm{nwt}\,|\,\mathscr{G}$. Any Eulerian tour traverses the cycle $u,w_1,w_2,v_{\min},v_{\max},w_4,u$ as a closed sub-tour. Hence, G_2 and G^* are equivalent 2DEE instances. □

5 Further remarks

We introduced the concept of interpreting sequencing problems as Eulerian Extension Problems. This view does not only lead to elegant and fast algorithms, it also allows an implicit representation of all optimal solutions for particular problem classes. We believe that our technique influences also other algorithmic frameworks for related problems, and moreover, it raises interesting potential for multi-criteria optimization.

As a first step towards multi-criteria optimization, consider both objective functions, $\mathscr{G}$ and $C_{\max}$. Already simple examples (even on two stages) show that schedules with optimal makespan may have the maximum number of interruptions whereas there exist idle time free schedules. For the two-stage variants of these problems, our technique provides us an implicit representation of all optimal solutions from which one could choose accordingly. Referring to a full version of this paper, we only mention here, that an algorithm that solves the interruption problem optimally, yields a schedule of makespan no greater than twice the minimum makespan, giving a trivial $(1,2)$-approximation for the bicriteria objective $(\mathscr{G}, C_{\max})$. Moreover, it follows from the *NP*-completeness of $\mathrm{F2}\,|\,\mathrm{nwt}, m_2 \geq 2\,|\,C_{\max}$ [15] that approximating such problems with a $(c,1)$-guarantee is *NP*-hard for any constant $c \geq 1$.

References

1. M. R. Garey and D. S. Johnson. *Computers and intractability*. Freeman, 1979.
2. K. Giaro. NP-hardness of compact scheduling in simplified open shop and flowshop. *European Journal of Operational Research*, 130:90–98, 2001.
3. K. Giaro and M. Kubale. Compact scheduling of zero-one time operations in multi-stage systems. *Discrete Applied Mathematics*, 145:95–103, 2004.
4. P. C. Gilmore and R. E. Gomory. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12:655–679, 1964.
5. 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. *Annals of Discrete Mathematics*, 5:287–326, 1979.
6. N. G. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3):510–525, 1996.
7. I. Harjunkoski and I. Grossmann. A decomposition approach for the scheduling of a steel plant production. *Computers and Chemical Engineering*, 25:1647–1660, 2001.
8. R. M. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
9. E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, 1985.
10. D. Pacciarelli and M. Pranzo. Production scheduling in a steelmaking-continuous casting plant. *Computers and Chemical Engineering*, 28(12):2823–2835, 2004.
11. S. S. Reddi and C. V. Ramamoorthy. On the flow-shop sequencing problem with no wait in process. *Operational Research Quarterly*, 23(3):323–331, 1972.
12. H. Röck. The three-machine no-wait flow shop is NP-complete. *Journal of the Association for Computing Machinery*, 31(2):336–345, 1984.
13. G. Rote and G. J. Woeginger. Time complexity and linear-time approximation of the ancient two-machine flow shop. *Journal of Scheduling*, 1(3):149–155, 1998.
14. C. Schwindt and N. Trautmann. Scheduling the production of rolling ingots: industrial context, model, and solution method. *Int. Trans. Operational Research*, 10(6):547–563, 2003.
15. C. Sriskandarajah and P. Ladet. Some no-wait shops scheduling problems: complexity aspect. *European Journal of Operational Research*, 24(3):424–438, 1986.
16. G. L. Vairaktarakis. Simple algorithms for gilmore–gomory's traveling salesman and related problems. *J. of Scheduling*, 6(6):499–520, 2003.
17. Z. Wang, W. Xing, and F. Bai. No-wait flexible flowshop scheduling with no-idle machines. *Operations Research Letters*, 33:609–614, 2005.
18. D. B. West. *Introduction to Graph Theory*. Prentice-Hall, 2nd edition, 2001.

A Appendix

Proof (of Lemma 3). We prove the statement by showing a more general result.

$$\begin{aligned}
&\operatorname{indeg}(V') - \operatorname{outdeg}(V') \\
&= \left(\sum_{v \in V'} \operatorname{indeg}(v) - |\{(u,v) \mid u,v \in V\}| \right) - \left(\sum_{v \in V'} \operatorname{outdeg}(v) - |\{(v,u) \mid u,v \in V\}| \right) \\
&= \sum_{v \in V'} \operatorname{indeg}(v) - \sum_{v \in V'} \operatorname{outdeg}(v)
\end{aligned}$$

□

A.1 Proofs for Section 3

Proof (of Lemma 5). Non-maximal up edges (u,v) are can be replaced with the edges $(u',v'),(v',v),(u,u')$, where (u',v') is the maximal up edge. The two other new edges point downwards, so the total number of up edges is preserved. As soon as all up edges are maximal, non-minimal downward edges can be replaced with minimal ones as described in the proof of Lemma 1. □

Proof (of Theorem 3). An instance $I' = (V,E)$ of $\mathscr{G}$-1DEE is constructed from an F2 | nwt | $\mathscr{G}$ instance I by defining $V = \bigcup_{i=1}^{n} \{p_{1i}, p_{2i}\}$ and by adding a *job edge* (p_{1i}, p_{2i}) to E for each job J_i, $i = 1,\ldots,n$.

In the following we show that the optimal solution to I' contains k up edges if and only if the optimal solutions to I cause $\max\{k-1,0\}$ interruptions.

Consider a schedule for I. For simplicity, we assume that the jobs are scheduled in the order of their index. We add an extension edge (p_{2i}, p_{1i+1}) to E' for $i = 1,\ldots,n-1$. This way we obtain a number of up edges equal to the number of interruptions caused by the schedule. Let (u,v) be the maximal up edge. Adding the edges $(p_{2n},u),(u,v),(v,p_{11})$ two of which are down edges, we obtain an extension admitting the Eulerian cycle $p_{11},p_{12},p_{21},p_{22},\ldots,p_{2n},u,v,p_{11}$.

Consider an optimal solution E' to I'. We employ Lemma 5 assuming that all down edges are minimal and all up edges are maximal. The set E' is converted into a solution to I by scheduling the jobs in the order the corresponding job edges appear in some Eulerian cycle in $(V, E \cup E')$.

If E' contains $k \geq 1$ up edges, then the schedule is started with the job edge appearing after one of them in the cycle. In the consequence, that specific up edge does not account for an interruption in the schedule. Whenever an interruption appears between two consecutive jobs, there is at least one other up edge between the corresponding job edges in the cycle. There is also at most one up edge between two consecutive job edges as otherwise up edges could be removed from the extension together with a sequence of down edges. Therefore, the number of interruptions is $k-1$.

If E' does not contain any up edge, then the schedule causes no interruption no matter which job it starts with. □

Proof (of Theorem 5). Given an instance I of problem FF2 | nwt | $\mathscr{G}$ with $m_1 = 1$, we consider an instance I' which equals I restricted to single machines on both stages, i.e., $m'_1 = m'_2 = 1$. We find an optimal solution S' for I' with r' interruptions. This can

be done efficiently by Theorem 4 and gives a feasible solution for I. Now, we construct an improved feasible schedule S for instance I with $r = \max\{0, r' - m_2 + 1\}$ interruptions: if there is an interruption in S' then we move the next block of interruption-free processing jobs to an unused machine of the second stage; we repeat until all interruptions are resolved or until all m_2 machines are used in S. This reduces the number of interruptions by $m_2 - 1$ or less if $r' < m_2 - 1$.

The solution S is optimal for I. To see that, assume for the sake of contradiction there is an optimal solution S^* with less interruptions $r^* < r$. Then the corresponding schedule can be transformed into a feasible one for instance I' with $r'' < r'$ interruptions. Run the set of jobs using machine m_i in S^* consecutively for $i = 1, \ldots, r^* - 1$ using only one processor at the second stage. This gives a feasible solution S'' for I' with at most $m_2 - 1$ interruptions more, i.e., $r'' \leq r^* + (m_2 - 1) < r + m_2 - 1$. This contradicts the optimality of schedule S' for I' with $r' \geq r + m_2 - 1$ interruptions. □

A.2 Proofs for Section 4

Proof (of Theorem 7). We provide a reduction from the *Three-Dimensional Matching Problem* (3DM): Given a set $U \subseteq M_1 \times M_2 \times M_3$ of triples, where M_1, M_2 and M_3 are pairwise disjoint and have the same number k of elements, decide whether U contains a subset $U' \subseteq U$ with $|U'| = k$ and no two elements of U' agree in any coordinate. Here we assume w.l.o.g. that any element of $M_1 \cup M_2 \cup M_3$ appears in at least one triple of U. The problem 3DM is well-known to be strongly *NP*-complete [8].

Denote the edges in a solution E' of 2DEE as *extension edges*. Note, that in contrast to the one-dimensional case, extension edges in this setting only contain down edges. We say that two points $u, v \in \mathbb{R}_0^+ \times \mathbb{R}_0^+$ are *independent*, if neither (u, v) nor (v, u) can be an extension edge due to the constraint specified in Definition 3.

Consider two rectangles $A = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}], A' = [x'_{\min}, x'_{\max}] \times [y'_{\min}, y'_{\max}]$ in $\mathbb{R}_0^+ \times \mathbb{R}_0^+$. We say that A and A' are *independent* if any two points $u \in A$, $v \in A'$ are independent. Formally, A and A' are independent if and only if either $x_{\max} < x'_{\min}$ and $y_{\min} > y'_{\max}$, or $x_{\min} > x'_{\max}$ and $y_{\max} < y'_{\min}$.

A point $v = (v_x, v_y)$ is *reachable* from another point $u = (u_x, u_y)$ if (u, v) is a down edge, and v is *one-way reachable* from u if it is reachable from u, but u is not reachable from v. Formally, v is reachable from u if $u_x \geq v_x$ and $u_y \geq v_y$. One-way reachability is obtained by additionally demanding $u \neq v$.

Given an instance $U \subseteq M_1 \times M_2 \times M_3$ of 3DM with $|M_1| = |M_2| = |M_3| = k$, we construct an equivalent 2DEE instance (V, E) as follows: Let $\{A_1, \ldots, A_{|U|}\}$ be a collection of pairwise independent rectangles. For $i = 1, \ldots, |U|$, define three points $a_{i1}, a_{i2}, a_{i3} \in A_i$ such that a_{i2} is one-way reachable from a_{i1}, and a_{i3} is one-way reachable from a_{i2}. We define the set of vertices as $V = \bigcup_{i=1,\ldots,|U|} \{a_{i1}, a_{i2}, a_{i3}\} \cup \{v_{\min}, v_{\max}\}$, where $v_{\min}$ is such that it is one-way reachable from any other vertex in V, and $v_{\max}$ is such that any other vertex in V is one-way reachable from it. Formally, the vertex set can be implemented as $v_{\min} = (0, 0)$, $v_{\max} = (|U| + 1, |U| + 1)$, and $a_{ij} = (4i - j, 4(|U| + 1 - i) - j)$ for $1 \leq i \leq |U|$ and $j \in \{1, 2, 3\}$.

For constructing the edge set E, let $U = \{U_1, \ldots, U_{|U|}\}$ be an arbitrary enumeration of the triples in U. For $j = 1, 2, 3$ and any element $b \in M_j$, we add edges such that (V, E) contains a directed cycle. That cycle, called C_b, includes exactly all vertices a_{ij} where b

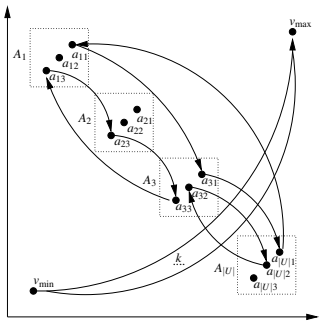

Fig. 1. 2DEE representation of a 3DM instance.

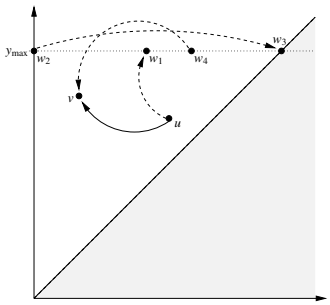

Fig. 2. The edge (u,v) is replaced with the legal edges (u,w_1), (w_2,w_3), and (w_4,v).

is the jth component of U_i. Consequently, E contains $3k$ pairwise vertex-disjoint cycles. The construction of E is completed by adding k edges from $v_{\min}$ to $v_{\max}$, see Fig. 1.

In the following, we assume that E' is a solution to the 2DEE problem instance (V,E). A Eulerian tour $(V,E\cup E')$ traverses $(v_{\min},v_{\max})$ exactly k times. The following two lemmata state crucial properties of such a tour.

Lemma 8. *Let $P = v_{\max},\ldots,v_{\min}$ be a path in $(V,E\cup E')$ that is part of a Eulerian tour and does not include the edge $(v_{\min},v_{\max})$. Then all edges in $P\cap E$ are contained in not more than three different cycles C_{b_1}, C_{b_2} and C_{b_2}.*

Lemma 9. *Any Eulerian tour in $(V,E\cup E')$ includes each cycle C_b as a contiguous sub-tour.*

Thus, a Eulerian tour leaves each cycle C_b at the same vertex it entered it. It follows that between any two consecutive traversals of $(v_{\min},v_{\max})$, three cycles C_{b_1},C_{b_2},C_{b_3} are traversed, each time starting and ending inside the same rectangle A_i. So the Eulerian extension of (V,E) has to have the form $E' = \bigcup_{h=1,\ldots,k}\{(v_{\max},a_{i_h1}),(a_{i_h1},a_{i_h2}),(a_{i_h2},a_{i_h3}),(a_{i_h3},v_{\min})\}$, where $i_1,\ldots,i_k$ are such that each cycle C_b can be traversed. This is the case if and only if no two a_{i_hj} and $a_{i_{h'}j}$ belong to the same cycle, which is equivalent to $U_{i_1},\ldots,U_{i_k}$ constituting a matching. □

Proof (of Lemma 8). Consider some edge in P that belongs to a cycle C_b with $b\in M_j$. There are only three possibilities to continue P after the sink a_{ij} of the edge has been reached. If P continues using another edge from E, then, due to the vertex-disjointness of the cycles, that edge also belongs to C_b. If P continues with an edge from E', then, due to the independence and reachability properties of V, that next edge will be either $(a_{ij},v_{\min})$ or $(a_{ij},a_{ij'})$ with $j'>j$. In the former case, P ends. In the latter case, P can enter a new cycle $C_{b'}$ with $b'\in M_{j'}$, but j' is strictly larger than j. In other words, each time P enters a new cycle C_b, $b\in M_j$, the index j strictly increases. As $j\leq 3$, the claim follows. □

Proof (of Lemma 9). As there are k edges $(v_{\min},v_{\max})$ and $3k$ cycles C_b, it follows from Lemma 8 that each cycle can be entered at most once, because otherwise some cycles

would remain not entered at all. Thus, in a Eulerian tour each cycle must be completely traversed as soon at it is entered. □

A.3 More complexity results for minimizing the number of interruptions (Section 4)

First we consider flexible flowshops with two machine stages. We show by reduction from 3-PARTITION that minimizing the number of interruptions is strongly *NP*-hard if the first stage contains two machines and the second stage only one.

Definition 4 (3-**PARTITION**). *Given a set A of $3m$ elements from $\mathbb{N}^+$ with $B/4 < a < B/2$ for all $a \in A$, where $B := \frac{1}{m}\sum_{a\in A} a$, decide whether A can be partitioned into m disjoint sets $A_1,\ldots,A_m$ with $\sum_{a\in A_i} a = B$ for $i = 1,\ldots,m$.*

It is well known that 3-PARTITION is *NP*-complete in the strong sense, see SP15 in [1].

Theorem 9. *The problem $\mathscr{E}_0(\mathrm{FF2}\,|\,\mathrm{nwt}\,|\,\mathscr{G})$ with numbers of machines $m_1 = 2$ and $m_2 = 1$ is strongly NP-complete.*

Proof. Given a $3m$-element instance A of 3-PARTITION, we construct an instance I of $\mathscr{E}_0(\mathrm{FF2}\,|\,\mathrm{nwt}\,|\,\mathscr{G})$ with $m_1 = 2$ and $m_2 = 1$. For any $a \in A$ we choose a job in I with processing times 1 and a on the first and second stage, respectively. To partition these jobs, we add $m+1$ auxiliary jobs having processing times B on the first stage and 0 on the second stage.

We show that in a schedule without interruptions, there is no point in time at which two auxiliary jobs are processed in parallel on the first stage. First notice, that two auxiliary jobs running fully in parallel must cause an interruption (i) with any job scheduled before them, because no job in I has processing time B on the second stage, and (ii) with any job scheduled after them, because no job in I has processing time 0 on the first stage. Now, assume that there are auxiliary jobs J_1 and J_2 with start times $S_1 < S_2 < S_1 + B$, which, as a consequence, fully block the first stage during $[S_2, S_1 + B)$. Since all jobs in I have positive processing times on the first stage, no job can start at $S_1 + B$ on the second stage, and thus, an idle time after J_1 is unavoidable.

Therefore, we may assume that in any schedule without interruptions all auxiliary jobs are processed on the same machine in the first stage. This induces m gaps of length at least B between the auxiliary jobs on the second stage. These gaps can be filled with the remaining jobs if and only if A is a yes-instance. □

Now, we generalize this *NP*-completeness result and combine it with the single machine case in Theorem 8. The following two lemmata show that $\mathscr{E}_0(\mathrm{FF}s\,|\,\mathrm{nwt}\,|\,\mathscr{G})$ with any constant number of stages $s \geq 2$ and constant numbers of machines, except $s = 2$ and $m_1 = 1$, is also strongly *NP*-complete.

Lemma 10. *Consider $\mathscr{E}_0(\mathrm{FF}s\,|\,\mathrm{nwt}\,|\,\mathscr{G})$ with a constant number of stages $s \geq 2$. Then any variant of this problem with constant numbers of machines $m_1,\ldots,m_s$ where $m_s = 1$ can be reduced to any other problem variant with constant numbers of machines m'_i, $i = 1,\ldots,s$, satisfying $m'_i \geq m_i$, $m'_s = 1$, and $m'_k > m_k$ for some $k < s$.*

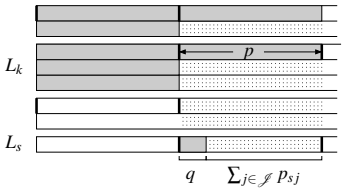

Fig. 3. Arrangement of auxiliary jobs in a schedule without interruptions (Lem. 10). Original jobs have to be processed in dotted slots.

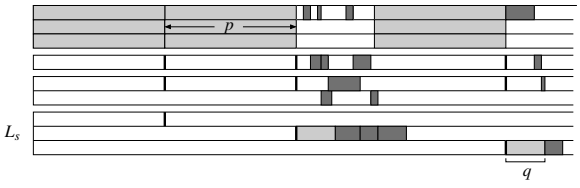

Fig. 4. Schedule without interruptions for I'. (Lem. 11)

Proof. We consider $\mathscr{E}_0(\mathrm{FF}s\,|\,\mathrm{nwt}\,|\,\mathscr{G})$ with a constant number of stages $s \geq 2$ and constant numbers of machines. Given an instance I of a problem variant with numbers of machines m_i, $i = 1,\ldots,s$, where $m_s = 1$, we build an instance I' of a problem variant with numbers of machines m'_i, satisfying $m'_i \geq m_i$, $m'_s = 1$, and $m'_k > m_k$ for some $k < s$. In addition to the original jobs $\mathscr{J}$ of I, we choose auxiliary jobs for I' which force the jobs from $\mathscr{J}$ to use only m_i machines on stage L_i. For blocking a stage L_i with $m'_i > m_i$ accordingly, we add $2m'_i - m_i$ jobs with processing time $p = \max_{j\in\mathscr{J}} \sum_{i=1}^{s-1} p_{ij} + \sum_{j\in\mathscr{J}} p_{sj}$ on stage L_i, and 0 elsewhere. We denote the auxiliary jobs corresponding to L_i by $\mathscr{J}_i$. For an arbitrary auxiliary job $J_{\mathrm{mod}} \in \mathscr{J}_k$, we modify the processing time on the last stage to $q = p - \sum_{j\in\mathscr{J}} p_{sj}$.

Given a schedule without interruptions for I', we consider the sub-schedule of auxiliary jobs. Observe, that the idle times on the last stage in this sub-schedule sum up to at most $\sum_{j\in\mathscr{J}} p_{sj}$, since otherwise they could not be filled with the jobs $\mathscr{J}$ in the total schedule. We examine the arrangement of jobs from $\mathscr{J}_k$. By its cardinality, there is a machine in stage L_k that processes a subset $\tilde{\mathscr{J}}_k \subseteq \mathscr{J}_k$ of at least two jobs. Due to the largest possible length of idle times on the only machine of the last stage, $\tilde{\mathscr{J}}_k$ does not contain more than two jobs. Thus, the jobs from $\tilde{\mathscr{J}}_k$ generate an idle time of length exactly p on the last stage. Since no further idle times are allowed, the remaining jobs from $\mathscr{J}_k$ and the other auxiliary jobs must be scheduled such that they exactly meet the previous jobs on the last stage. In particular, the job J_{mod} must be scheduled such that it is processed at the beginning of the idle time, produced by the two jobs from $\tilde{\mathscr{J}}_k$ that are processed on the same machine on L_s. Thus, we can assume that the auxiliary jobs are arranged as in Fig. 3.

By definition of p and q there exists a schedule without interruptions for the instance I' if and only there exists one for the instance I. □

Lemma 11. *Consider $\mathscr{E}_0(\mathrm{FF}s\,|\,\mathrm{nwt}\,|\,\mathscr{G})$ with a constant number of stages $s \geq 2$. Then any variant of this problem with constant numbers of machines $m_1,\ldots,m_s$ can be reduced to any other problem variant with constant numbers of machines m'_i, $i = 1,\ldots,s$, satisfying $m'_s \geq m_s$ and $m'_i = m_i$ for $i = 1,\ldots,s-1$.*

Proof. We reduce the problem $\mathscr{E}_0(\mathrm{FF}s\,|\,\mathrm{nwt}\,|\,\mathscr{G})$ with constant numbers of machines $m_1,\ldots,m_s$ to another variant of this problem with machine numbers m'_i, $i = 1,\ldots,s$, satisfying $m'_s \geq m_s$ and $m'_i = m_i$ elsewhere. Given an instance I of the former problem

with set of jobs $\mathscr{J}$, we build an instance I' of the latter problem by adding auxiliary jobs to the instance I: We choose m_s jobs with processing time $q = \max_{j\in\mathscr{J}} \sum_{i=1}^{s} p_{ij}$ on the last stage, and 0 on all other stages. Furthermore, we add $m'_1 m'_s$ jobs with processing time $p > (m_s+1)\,q + \sum_{j\in\mathscr{J}} p_{sj}$ on stage L_1, and 0 elsewhere. We denote these jobs with $\mathscr{J}^q_{\text{aux}}$ and $\mathscr{J}^p_{\text{aux}}$, respectively.

If there exists an interruption-free schedule S for I, then the schedule in Fig. 4 shows how to arrange the jobs from I' without idle times on the last stages: The jobs from $\mathscr{J}^p_{\text{aux}}$ are scheduled in blocks of m'_1 jobs which are processed in parallel on the first stage and on the same machine in the last stage. With sufficiently long idle times between these blocks, we can process after m_s of them first a job from $\mathscr{J}^q_{\text{aux}}$ and then a sub-schedule that corresponds to a last stage machine in S.

On the other hand, given a schedule for I' without interruptions, we can construct a schedule without interruptions for I. In the following we call jobs with processing time 0 on the stages $L_1,\ldots,L_{s-1}$ *zero-jobs*. We claim that in any interruption-free schedule for I' every machine on the last stage processes first one or more jobs from $\mathscr{J}^p_{\text{aux}}$, second a zero-job and third an interruption-free sequence of jobs from $\mathscr{J}\cup\mathscr{J}^q_{\text{aux}}$ (which may also include jobs from $\mathscr{J}^p_{\text{aux}}$).

We consider a schedule for I' that does not contain an interruption. Then, the last stage idle times in the sub-schedule induced by the jobs $\mathscr{J}^p_{\text{aux}}$ do not equal or exceed p, by definition of p. Thus, there is no pair of jobs from $\mathscr{J}^p_{\text{aux}}$ that is processed on the same machine on the first and last stage. In particular, every first stage machine processes exactly m'_s jobs from $\mathscr{J}^p_{\text{aux}}$. Hence, given a machine on the first and last stage, there is a unique job from $\mathscr{J}^p_{\text{aux}}$ which is processed on both machines. As a consequence, the m'_1 jobs from $\mathscr{J}^p_{\text{aux}}$, that are processed on a particular machine M on the last stage, use every machine in the first stage for processing exactly one job.

Let c be the first completion time of these jobs, and let $J\in\mathscr{J}^p_{\text{aux}}$ be a job completing at that time. Since idle times on the last stage must be compensable with the jobs from I', none of the jobs from $\mathscr{J}^p_{\text{aux}}$ on M completes after $d = c + m_s q + \sum_{j\in\mathscr{J}} p_{sj}$. Thus, these jobs block the first stage during $[d-p,c)$. This yields two characteristics for schedules without interruptions: First, J can be followed on machine M only by zero-jobs. And second, by

$$\big|[d-p,c)\big| \;=\; p - m_s q + \sum_{j\in\mathscr{J}} p_{sj} \;>\; q = \max_{j\in\mathscr{J}} \sum_{i=1}^{s} p_{ij},$$

no job can be processed before J on M. Since the jobs from $\mathscr{J}^p_{\text{aux}}$ have processing time 0 on the last stage, all further jobs from $\mathscr{J}\cup\mathscr{J}^q_{\text{aux}}$ on M follow the zero-job without idle time. This completes the proof of the claim.

Given a schedule for I' with the properties claimed above, we show how to construct an interruption-free schedule for the instance I. Consider all machines on the last stage that process a job from $\mathscr{J}\cup\mathscr{J}^q_{\text{aux}}$. If there is a machine that processes more than one job from $\mathscr{J}^q_{\text{aux}}$, then we move the sequence of jobs, starting with the second job from $\mathscr{J}^q_{\text{aux}}$ (ignoring the jobs from $\mathscr{J}^p_{\text{aux}}$), to a machine on the last stage, that processes no job from $\mathscr{J}^q_{\text{aux}}$. Since the jobs from $\mathscr{J}^q_{\text{aux}}$ are zero-jobs, this does not create an interruption on the new machine. If there occur conflicts with other jobs, then we delay sub-schedules corresponding to some machines on the last stage. In case that a machine

processes only jobs from $\mathscr{J}$, but no job from $\mathscr{J}^q_{\text{aux}}$, then by our claim, the first job is a zero-job. Hence, we can move the sequence of jobs from $\mathscr{J}$ to another machine, which already processes a job from $\mathscr{J}^q_{\text{aux}}$. Thus, we may assume, that there are at most m_s machines on the last stage that process jobs from $\mathscr{J}$ and each of these machines processes one job from $\mathscr{J}^q_{\text{aux}}$. If the job from $\mathscr{J}^q_{\text{aux}}$ is not the first one in the sequence of jobs from $\mathscr{J} \cup \mathscr{J}^q_{\text{aux}}$, then we can move the subsequence starting with it to the beginning. Since the first job in the sequence and the job from $\mathscr{J}^q_{\text{aux}}$ are zero jobs, this creates no interruption. Finally, we obtain a schedule for I', that uses only m_s machines on the last stage, and in which the jobs of $\mathscr{J}$ are processed without interruption. This yields an interruption-free schedule for the instance I. □

Theorem 6 is proved by combining the results of this subsection with Theorem 8.