Equitable Scheduling on a Single Machine

**Klaus Heeger · Danny Hermelin · George B. Mertzios · Hendrik Molter ·
Rolf Niedermeier · Dvir Shabtay**

Abstract We introduce a natural but seemingly yet unstudied variant of the problem of scheduling jobs on a single machine so as to minimize the number of tardy jobs. The novelty of our new variant lies in simultaneously considering several instances of the problem at once. In particular, we have n clients over a period of m days, where each client has a single job with its own processing time and deadline per day. Our goal is to provide a schedule for each of the m days, so that each client is guaranteed to have their job meet its deadline in at least $k \leq m$ days. This corresponds to an equitable schedule where each client is guaranteed a minimal level of service throughout the period of m days. We provide a thorough analysis of the computational complexity of three main variants of this problem, identifying both efficient algorithms and worst-case intractability results.

Keywords Resource Allocation · Fairness · Equity · Fixed-Parameter Tractability · Approximation

K. Heeger, R. Niedermeier
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Berlin, Germany
E-mail: heeger@tu-berlin.de, rolf.niedermeier@tu-berlin.de

D. Hermelin, H. Molter, D. Shabtay
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel
E-mail: hermelin@bgu.ac.il, molter@post.bgu.ac.il, dvirs@bgu.ac.il

G. B. Mertzios
Department of Computer Science, Durham University, Durham, UK
E-mail: george.mertzios@durham.ac.uk

1 Introduction

One of the most basic and fundamental scheduling problems is that of minimizing the number of tardy jobs on a single machine. In this problem we are given n jobs, where each job j has an integer processing time p_j and an integer deadline d_j, and the goal is to find a permutation of the jobs so that the number of jobs exceeding their deadlines is minimized (a job j exceeds its deadline if the total processing time of jobs preceding it in the schedule, including itself, is larger than d_j). This problem is known as the $1||\sum U_j$ problem in the classical three-field notation for scheduling problems by Graham et al. [26]. It is well-known that $1||\sum U_j$ is solvable in $O(n \log n)$ time [45, 48, 54], but becomes NP-hard in case of simple (chain) precedence constraints even if all processing times p_j are the same [42]. There is also a more recent survey concerning the minimization of the weighted number of tardy jobs [1], and the problem has also been thoroughly studied for parallel machines [6].

Due to the ever increasing importance of high customer satisfaction, fairness-related issues are becoming more and more important in all areas of resource allo-

cation [12, 22, 39, 40, 56][1]. For instance, in their seminal work Baruah et al. [7] introduced the concept of proportionate progress, a fairness concept for resource allocation problems. Nowadays, equity and fairness in resource allocation is a widely discussed topic, leading to considerations such as the "price of fairness" [10], or to discussions about the abundance of fairness metrics [27].

We study a natural yet seemingly novel variant of the $1||\sum U_j$ problem which takes into account a very basic aspect of equity among the customers. In our model, we have m instances of $1||\sum U_j$ which represent scheduling requests of a set of n clients over a period of m days. As this is an initial study, we assume all instances are known in advance, and focus solely on the offline setting. Additionally, we receive as input a non-negative integer $k \leq m$ representing the *equity parameter*. Our goal is provide a schedule for each of the m days that ensures an equitable service among the clients. More specifically, we want to ensure that the jobs of each client are non-tardy in at least k out of the m days. In this way, the number of days different clients receive satisfactory service differs by at most $m - k$, and so the larger k is, the more equitable our solution becomes.

1.1 Formal Model

Our model can be formally described as follows: We wish to schedule non-preemptively the jobs of a set of n clients over a period of m days in an equitable way. At each day, each client has a single job to be scheduled non-preemptively on a single machine. We let $p_{i,j}$ and $d_{i,j}$ respectively denote the integer *processing time* and *deadline* of the job of client $i \in \{1, \ldots, n\}$ at day $j \in \{1, \ldots, m\}$. In addition, we let k denote an *equity parameter* given as input, with $k \in \{0, \ldots, m\}$.

A schedule σ_j for day $j \in \{1, \ldots, m\}$ is a permutation $\sigma_j : \{1, \ldots, n\} \to \{1, \ldots, n\}$ representing the order of jobs to be processed on our single machine on day j. For a given schedule σ_j, the *completion time* $C_{i,j}$ of the job of client i is defined as $C_{i,j} = \sum_{\sigma_j(i_0) \leq \sigma_j(i)} p_{i_0,j}$. In this way, the job meets its deadline on day j if $C_{i,j} \leq d_{i,j}$. If this is indeed the case, then we say that client i is *satisfied* on day j, and otherwise i is *unsatisfied*. Our goal is to ensure that each client is satisfied in at least k days out of the entire period of m days; such a solution schedule (for all m days) is referred to as *k-equitable*. Thus, depending on how large k is in

comparison with m, we ensure that no client gets significantly worse service than any other client.

> EQUITABLE SCHEDULING (ES):
>
> Input: A set of n clients, each having a job with processing time $p_{i,j}$ and deadline $d_{i,j}$ for each day $j \in \{1, \ldots, m\}$, and an integer k.
> Task: Find a set of m schedules $\{\sigma_1, \ldots, \sigma_m\}$ so that for each $i \in \{1, \ldots, n\}$ we have $|\{j \mid 1 \leq j \leq m \wedge C_{i,j} \leq d_{i,j}\}| \geq k$.

1.2 Three Equitable Scheduling Variants

As EQUITABLE SCHEDULING turn to be strongly NP-hard even for very restricted special cases (see Section 1.5), we decided to look on special cases which are commonly studied in the scheduling literature.

- In the first variant, which we call EQUITABLE SCHEDULING WITH UNIT PROCESSING TIMES (ESUP), the processing time of all jobs are unit in each day. That is, we have $p_{i,j} = 1$ for each $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$.
- In the EQUITABLE SCHEDULING WITH SINGLE DEADLINES (ESSD) problem, all jobs have the same deadline at each day. That is, at each day $j \in \{1, \ldots, m\}$ we have $d_{i,j} = d_j$ for each $i \in \{1, \ldots, n\}$.
- In the final variant, called EQUITABLE SCHEDULING WITH PRECEDENCE CONSTRAINTS (ESPC), all processing times are unit, and jobs share the same deadline at each day, i.e., $p_{i,j} = 1$ and $d_{i,j} = d_j$ for each $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. In addition, in each day we are given a *precedence* directed acyclic graph (DAG) $G_j = (\{1, \ldots, n\}, E_j)$ which represents precedence constraints on the jobs at day j. We say that a schedule σ_j is *feasible* if for each $(i_1, i_2) \in E_j$ we have $\sigma_j(i_1) < \sigma_j(i_2)$.

Note that equal processing time is one of the most widely studied special cases in the scheduling literature (see *e.g.* [3, 4, 5, 13, 14, 50, 53, 57]). Such models typically deal with real-world applications of repetitive manufacturing of the same product to satisfy different orders. The single deadline case is also very common in real-world applications, as it captures cases where there is a single shipment for all jobs during the scheduling horizon, and/or cases where jobs are assigned the same due-date in order to treat costumers equally (*e.g.* [2, 11, 41, 49, 52, 55]). Finally, precedence constraints are widely studied in the area of scheduling (*e.g.* [15, 16, 19, 33, 43, 47]), typically in scenarios such as engineering, where machines are assembled or disassembled, logistics, or project management.

---

[1] In 2018, ACM started its new conference series on "Fairness, Accountability, and Transparency (originally FAT, since 2021 FAccT)".

For each of the three variants above, we will also consider the special case where the input for each day is the same, and we will append an '*' to the name of the problem variant to indicate that this is the case we are considering. For example, in the ESPC* problem we have $d_{j_1} = d_{j_2}$ and $G_{j_1} = G_{j_2}$ for all $j_1, j_2 \in \{1, \ldots, m\}$.

1.3 Example Applications

Below we discuss three example applications of our equitable scheduling model. The goal of these examples is not only to illustrate real-world scenarios where our model can be useful, but also to better illuminate the three variants discussed above.

As a first example, consider the following patient scheduling problem: There is a set of n patients that must be treated by a single machine in a clinic, say a dialysis machine. The machine works on daily basis and each of the patients must receive a daily treatment throughout m days. The daily schedule begins at 7:00 AM, and each of the patients has a different deadline in each day that he wishes to meet (corresponding to the public transportation time tables and constraints from his workplace). To ensure fairness among the patients, the clinic wishes to make sure that each of the patients will not be late to the last train or work in at least k of the days. This scenario can naturally be modeled by our EQUITABLE SCHEDULING problem. In case the treatment is in fact dialysis, typically each treatment takes four hours regardless of the patient, and the problem naturally becomes EQUITABLE SCHEDULING WITH UNIT PROCESSING TIMES.

Consider now a second example of a research group with n PhD students that have access to a single research lab. At the beginning of the semester, each student has to submit a plan for their experiments for the next m days, where each experiment requires the usage of the entire lab for a varying amount of time. In addition, due to space constraints (and possibly health regulations) at most one student can use the lab at any given time. All students request access to the lab each day, but due to high demand not all of them can be scheduled early enough before the lab closes for the day. The chair of the group wants to ensure fairness amongst the students, and so they are looking for a schedule where every student can perform their experiments in the lab on at least k out of the m days. This problem precisely corresponds to the EQUITABLE SCHEDULING WITH SINGLE DEADLINES problem, as all students have the same deadline for their experiments (the closing time of the lab). We remark that the authors of the paper faced a similar problem to the one described above when the second author was visiting TU Berlin.

As a final example, consider a factory manufacturing three products: A, B, and C. The time for manufacturing each product takes roughly the same time. However, due to the configuration process of the machines at the factory, all orders of product A must be manufactured first, followed by all orders of product B, and finally all orders of product C. The factory has n clients who place requests for the next m days. In the basic case, each client orders a single product every day (but the products they order may differ between different days), and all clients want their product shipped to them at the same time (say, before the end of the day). To ensure an equitable solution amongst its clients, the factory wishes to meet the deadlines of the clients in at least k out of the m days. This naturally corresponds to the EQUITABLE SCHEDULING WITH PRECEDENCE CONSTRAINTS problem, where the precedence constraints are determined by the manufacturing order of all products of the same day.

1.4 Methodology

We are mainly interested in theoretical analysis of the problems, focusing on complexity analysis of the problems in hand, and providing exact algorithms or algorithms with provable approximation guarantees.

We study the (parameterized) algorithmic complexity[2] of all three main variants (and some further variations) discussed above. We use the following concepts from parameterized complexity theory [17, 20, 21, 51]. A *parameterized problem* $L \subseteq \{(x, k) \in \Sigma^* \times \mathbb{N}\}$ is a subset of all instances (x, k) from $\Sigma^* \times \mathbb{N}$, where k denotes the *parameter*. A parameterized problem L is

- FPT (*fixed-parameter tractable*) if there is an algorithm that decides every instance (x, k) for L in $f(k) \cdot |x|^{O(1)}$ time, and
- contained in the class XP if there is an algorithm that decides every instance (x, k) for L in $|x|^{f(k)}$ time,

where f is any computable function that depends only on the parameter. If a parameterized problem L is W[1]-hard, then it is presumably not fixed-parameter tractable [17, 20, 21, 51].

We remark that any NP-hardness result for any variant of EQUITABLE SCHEDULING for a specific $k \in \{1, \ldots, m\}$ (or $m - k \in \{1, \ldots, m\}$), implies NP-hardness for all $k' > k$ (or $m - k' > m - k$). This can be seen by adding and additional $k' - k$ days in which

---

[2] Parameterized complexity studies of NP-hard scheduling problems gained more and more interest over the recent years [8, 9, 23, 30, 31, 32, 46]; we contribute to this field with several of our results.

all jobs are early under *any* schedule (or all jobs are late under *any* schedule), *e.g.* the common deadline on each day is larger than the total processing time of all jobs on that day (or the processing time of all jobs on that day is by one larger than the deadline). Consider now the special case of either of our problems where we have the same instance for each of the days *e.g.* ESSD* and ESPC*, and assume that it is NP-hard for $k = 1$ (or $m - k$). For proving that it is NP-hard for any $k' > 1$ (or $m - k' > m - k$) value we simply duplicate the reduction for $k = 1$ (or $m - k = 1$) by including additional $(k' - 1)m$ (or $(m - k' - 1)m$) days. Thus, in what follows, all NP-hardness results presented for $k = 1$ (or $m - k = 1$) actually apply for all values of k (or $m - k$). In the case of ESPC, all NP-hardness results for unit processing times also transfer to the case of equal but non-unit processing times. This can be shown by appropriately rescaling the processing times and deadlines in the NP-hardness reductions.

1.5 Our Results and Roadmap

We present several positive and negative results regarding the computational complexity of the three variants of EQUITABLE SCHEDULING discussed above. Our main findings in the paper are listed below.

- For ESUP, we show that the problem can be solved in polynomial time by devising a reduction to the BIPARTITE MAXIMUM MATCHING problem. Our reduction can also be applied when jobs have arbitrary integral release times and there is a fixed number of machines available on each day.
- For ESSD and ESSD*, we show strong NP-hardness and W[1]-hardness for parameter m (number of days) even if $k = 1$ or $m - k = 1$, and weak NP-hardness even if $m = 2$ and $k = 1$. On the positive side, we show that ESSD can be solved in pseudo-polynomial time for $m = O(1)$, and is in FPT when parameterized by the number n of clients. For ESSD*, we give a polynomial-time algorithm that for any k, computes a $2k$-equitable solution assuming there exists a $3k$-equitable solution.
- For ESPC, we show NP-hardness even if $k = 1$ and $m = 2$. For ESPC*, we show NP-hardness and W[1]-hardness for parameter m even if $k = 1$ or $m - k = 1$ and the precedence DAG only consists of disjoint paths. For ESPC, we also show NP-hardness for $k = 1$ if each precedence DAG either consists of a constant number of disjoint paths or disjoint paths of constant length. On the positive side, we show that ESPC is in FPT when parameterized by the number n of clients.

The remainder of the paper is organized according to the three variants of EQUITABLE SCHEDULING. All results for ESUP are presented in Section 2, Section 3 contains all results for ESSD and ESSD*, while Section 4 is concerned with ESPC and ESPC*. Finally, we discuss some open problems in Section 5.

2 Unit Processing Times

In this section, we show that EQUITABLE SCHEDULING WITH UNIT PROCESSING TIMES can be solved in polynomial time by a reduction to the BIPARTITE MAXIMUM MATCHING problem. Later in the section we will show that our reduction can also be applied when jobs have arbitrary integer release times, and there is a fixed number of machines available on each day.

Recall that $p_{i,j}$ and $d_{i,j}$ respectively denote the processing time (which is unit in this case, i.e., $p_{i,j} = 1$) and deadline of the job of client i on day j, and that k is the equity parameter. Let $d_j^* = \max_{1 \leq i \leq n} d_{i,j}$ denote the maximal deadline on day $j \in \{1, \ldots, m\}$. We create an undirected graph G with the following vertices:

- For each $i \in \{1, \ldots, n\}$ and each $j \in \{1, \ldots, m\}$, we create a vertex $v_{i,j}$. The set of vertices $V = \{v_{i,j} : 1 \leq i \leq n, 1 \leq j \leq m\}$ represents all input jobs of all clients.
- For each $d \in \{1, \ldots, d_j^*\}$ and each $j \in \{1, \ldots, m\}$, we create a vertex $u_{d,j}$. The set $U = \{u_{d,j} : 1 \leq d \leq d_j^*, 1 \leq j \leq m\}$ represents all possible completion times of the all input jobs that meet their deadline.
- For each $i \in \{1, \ldots, n\}$ and each $j \in \{1, \ldots, m - k\}$, we create a vertex $w_{i,j}$. The set $W = \{w_{i,j} : 1 \leq i \leq n, 1 \leq j \leq m - k\}$ represents the set of jobs that exceed their deadline.

The edges of G are constructed as follows. For each $i \in \{1, \ldots, n\}$ and each $j \in \{1, \ldots, m\}$ we connect $v_{i,j}$ to:

- vertices $w_{i,1}, \ldots, w_{i,m-k}$, and
- vertices $u_{1,j}, \ldots, u_{d,j}$, where $d = d_{i,j}$.

Lemma 1 *G has a matching of size nm if and only if there exists schedules $\{\sigma_1, \ldots, \sigma_m\}$ where no client is unsatisfied in more than $m - k$ days.*

Proof ($\Leftarrow$): Let $\{\sigma_1, \ldots, \sigma_m\}$ be a set of schedules where no client is unsatisfied on more than $m - k$ days. Consider the job of client i on day j, for some $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. Note that the completion time of this job is $C = C_{i,j} = \sigma_j(i)$. If $C \leq d_{i,j}$, then there is an edge $\{v_{i,j}, u_{C,j}\}$ in G, and we add this edge to the matching. If $C > d_{i,j}$, then client i is unsatisfied on day j. Let ℓ denote the number of days prior to j that

client i is unsatisfied. Then $\ell < m - k$, since otherwise client i would be unsatisfied in more than $m - k$ days (including day j). We add the edge $\{v_{i,j}, w_{i,\ell+1}\}$ to the matching. In total, this gives us a matching of size nm in G.

($\Rightarrow$): Assume that G contains a matching of size nm. We create a set of schedules $\{\sigma_1, \ldots, \sigma_m\}$ accordingly. First note that the fact that G is bipartite with one part being V, and $|V| = mn$, implies that every vertex in V has to be matched. Let $v_{i,j} \in V$. Then this vertex is either matched to a vertex in U or a vertex in W. Because $N(u_{d+1,j}) \subseteq N(u_{d,j})$ holds for every $j \in \{1, \ldots, m\}$ and $d \in \{1, \ldots, d_j^* - 1\}$, we may assume that for every $j \in \{1, \ldots, m\}$ there exists some d_j such that vertex $u_{d,j}$ is matched if and only if $d \leq d_j$.

– Suppose that $v_{i,j}$ is matched to some $u_{d,j_0} \in U$. Observe that $j = j_0$ and $d \leq d_{i,j}$ by construction of G. We set $\sigma_j(i) = d$, and so client i is satisfied on day j. Observe that the fact that u_{d,j_0} cannot be matched to any other vertex in V guarantees that $\sigma_j(i) \neq \sigma_j(i_0)$ for any $i_0 \neq i$. Let s_j denote the number of clients satisfied by σ_j in this way.
– Suppose that $v_{i,j}$ is matched to some vertex $w_{i_0,j_0} \in W$. Note that $i = i_0$ and $j_0 \leq m - k$ by construction of G. Let $x_{i,j} = |\{i_0 < i : w_{i_0,j}$ is matched $\}|$. Then we set $\sigma_j(i) = s_j + x_{i,j} + 1$.

After each σ_j is permutation from $\{1, \ldots, n\}$ to $\{1, \ldots, n\}$, and no client is unsatisfied in more than $m - k$ days under $\{\sigma_1, \ldots, \sigma_m\}$. $\quad\square$

Since we may assume that $d_j^* \leq n$ for every $j \in \{1, \ldots, m\}$, we observe that G has $O(mn)$ vertices and $O(mn^2 + m^2n)$ edges, and it can be constructed in $O(mn^2 + m^2n)$ time. Using the algorithm of Hopcroft and Karp [34] for BIPARTITE MAXIMUM MATCHING, this gives us the following:

Theorem 1 *ESUP can be solved in $O((n+m) \cdot (nm)^{\frac{3}{2}})$ time.*

We remark that the algorithm behind Theorem 1 is very flexible and can easily be extended to the settings where there are multiple parallel machines and jobs have either arbitrary integral release dates or equal processing time (which is not necessarily of unit length).

The main idea is to first modify the meaning of the vertices in U. In the first case (multiple parallel machines and arbitrary release dates) the vertices in U represent unit time slots for jobs and a job has an edge to all time slots that begins not earlier than the job's release date and finish no later then the job's deadline. In the second case (multiple parallel machines and equal processing times) the vertices in U represent time slots

of equal length for jobs, and a job has an edge to all time slots that ends before the job's deadline.

In both cases, we model parallel machines by essentially introducing copies of the vertices in U for each of the machines.

We next give a formal description on how to construct the graph G for the first case, but leave out the details for the second case (for the sake of brevity, and due to high level of similarity in the reduction).

Recall that $p_{i,j}$ and $d_{i,j}$ respectively denote the processing time and deadline of the job of client i on day j, and that k is the equity parameter. Let $r_{i,j} < d_{i,j}$ the release date of the job of client i on day j and x_j the number of parallel machines available on day j. Let $d_j^* = \max_{1 \leq i \leq n} d_{i,j}$ denote the maximal deadline on day $j \in \{1, \ldots, m\}$ and let $x^* = \max_{1 \leq j \leq m} x_j$ denote the maximal number of machines available on a day. We create an undirected graph G with the following vertices:

– For each $i \in \{1, \ldots, n\}$ and each $j \in \{1, \ldots, m\}$, we create a vertex $v_{i,j}$. The set of vertices $V = \{v_{i,j} : 1 \leq i \leq n, 1 \leq j \leq m\}$ represents all input jobs of all clients.
– For each $d \in \{1, \ldots, d_j^*\}$, each $j \in \{1, \ldots, m\}$, and each $x \in \{1, \ldots, x^*\}$, we create a vertex $u_{d,j,x}$. The set $U = \{u_{d,j,x} \mid 1 \leq d \leq d_j^*, 1 \leq j \leq m, 1 \leq x \leq x^\star\}$ represents all possible completion times on some machine of the all input jobs that meet their deadline.
– For each $i \in \{1, \ldots, n\}$ and each $j \in \{1, \ldots, m-k\}$, we create a vertex $w_{i,j}$. The set $W = \{w_{i,j} : 1 \leq i \leq n, 1 \leq j \leq m-k\}$ represents the set of jobs that exceed their deadline.

The edges of G are constructed as follows. For each $i \in \{1, \ldots, n\}$ and each $j \in \{1, \ldots, m\}$ we connect $v_{i,j}$ to:

– vertices $w_{i,1}, \ldots, w_{i,m-k}$, and
– vertices $u_{r,j,1}, \ldots, u_{d,j,x}$, where $d = d_{i,j}, r = r_{i,j}+1$, and $x = x_j$.

The proof of correctness is analogous to the proof of Lemma 1. We omit the details.

3 Single Deadline on Each Day

In this section, we investigate the computational complexity of EQUITABLE SCHEDULING WITH SINGLE DEADLINES.

3.1 Hardness Results

We first show that ESSD is NP-hard even if all numbers involved are small constants.

Theorem 2 ESSD *is NP-hard even if* $k = 1$ *and* $d = 3$.

Proof We present a polynomial-time many-one reduction from INDEPENDENT SET on 3-regular graphs. In this problem we are given a graph $G = (V, E)$ where every vertex has degree 3 and an integer ℓ, and we are then asked whether G contains an independent set of size ℓ. This problem is known to be NP-hard [25]. Let $(G = (V, E), \ell)$ be an instance of INDEPENDENT SET, where G is an 3-regular graph.

We construct an instance of ESSD as follows. We set $k = 1$ and $d = 3$. For each vertex $v \in V$, we add a client c_v, and for each edge $e \in E$, we add a client c_e. There are $m = 2|V| - \ell$ many days. We order the vertices arbitrarily, i.e., $V = \{v_1, \ldots, v_n\}$. On day i, the job of client c_{v_i} has processing time 3, while the jobs of client c_v with $v \neq v_i$ have processing time 4. The jobs of clients c_e have processing time 1 if e is incident to v_i and processing time 4 otherwise. Finally, on days $|V| + 1$ to $2|V| - \ell$, jobs of clients c_v for $v \in V$ have processing time 3, and jobs of clients c_e for $e \in E$ have processing time 4.

($\Rightarrow$): An independent set I of size ℓ implies a feasible schedule by scheduling the jobs of the $|V| - \ell$ clients c_v with $v \in V \setminus I$ to the days $|V| + 1$ to $2|V| - \ell$. For each $v_i \in I$, we schedule the job of client c_{v_i} to day i. For all clients c_e we schedule one of their jobs on a day corresponding to an arbitrary endpoint of e that is not contained in the independent set I.

($\Leftarrow$): Note that every feasible schedule must schedule a job of client c_{v_i} to day i for at least ℓ different clients c_{v_i}. We claim that $I := \{v_i \in V \mid c_{v_i}$ has a job that is scheduled on day $i\}$ is an independent set. If there exists an edge $e = \{v_i, v_{i'}\}$ for $v_i, v_{i'} \in I$, then this edge can be scheduled neither on a day i nor on day i' and therefore cannot be scheduled on any day. Thus, these at least ℓ vertices in I form an independent set. $\square$

Note that in ESSD w.l.o.g. the maximum processing times (and hence also the number of processing times) is upper-bounded by the deadline. Thus Theorem 2 also gives us also NP-hardness for constant maximum processing time.

We can further show that ESSD* (i.e., ESSD where the processing time of the job of each client is the same every day) is NP-hard and W[1]-hard when parameterized by the number of days.

Theorem 3 ESSD* *is NP-hard and W[1]-hard when parameterized by the number m of days even if $k = 1$ and all numbers are encoded in unary.*

Proof We present a parameterized reduction from UNARY BIN PACKING, where given a set $I = \{1, \ldots, n\}$ of items with sizes s_i for $i \in I$, b bins of size B, we are asked to decide whether it is possible to distribute the items to the bins such that no bin is overfull, i.e., the sum of the sizes of items put into the same bin does not exceed B. UNARY BIN PACKING is known to be NP-hard and W[1]-hard when parameterized by b [35]. Given an instance of UNARY BIN PACKING, we construct an instance of ESSD* as follows.

We set the number of days to b, i.e., $m = b$. For each item $i \in I$ and each day $j \in \{1, \ldots, m\}$ we create a job for client i with processing time $p_{i,j} = s_i$, i.e., the processing time is the same every day. The deadline for every day is $d = B$. Finally, we set $k = 1$. This finishes the construction which can clearly be done in polynomial time.

($\Rightarrow$): Assume the UNARY BIN PACKING instance is a YES-instance. Then there is a distribution of items to bins such that no bin is overfull. If item $i \in I$ is put into the jth bin for some $j \in \{1, \ldots, b\}$, then we schedule the job of of client i on day j to be processed. Since every item is put into one bin, every client has a job that is scheduled to be processed at one day and since no bin is overfull, all scheduled jobs can be processed before their deadline d. It follows that we have a 1-equitable set of schedules.

($\Leftarrow$): Assume we have a 1-equitable set of schedules. Then every client has a job on at least one day that is processed. Let client i have a job that is processed on day j. Then we put item i into the jth bin. Since the processing time $p_{i,j}$ is the same as the size of item i and the sum of the processing times of jobs that are scheduled to be processed on the same day is at most $d = B$, the sum of sizes of items that are put into the same bin is at most B. Hence, we have a valid distribution of items into bins.

We remark that the W[1]-hardness only transfers to all larger *constant* values of k by "duplicating" the reduction as described in Section 1.4, because the number of days is also increased by this process. $\square$

Note that the hardness result above applies for cases where the clients do not need to be satisfied most of the days. Next we show that with minor modifications to the reduction above, we can also obtain hardness for the case that $m - k \geq 1$.

Corollary 1 ESSD* *is NP-hard and W[1]-hard when parameterized by the number m of days even if $m - k = 1$ and all numbers are encoded in unary.*

Proof (sketch) We use the same reduction from UNARY BIN PACKING [35] as in the proof of Theorem 3 with some minor changes: We set the deadline for every day to $d = \sum_i s_i - B$. We furthermore assume w.l.o.g. that

$\sum_i s_i = b \cdot B$ (this can be achieved by adding a sufficient amount of size one items to the Unary Bin Packing instance). Lastly, we set $k = m - 1$.

($\Rightarrow$): Assume the Unary Bin Packing instance is a YES-instance. Then there is a distribution of items to bins such that no bin is overfull and since we assume $\sum_i s_i = b \cdot B$, we have that every bin is exactly filled. If item $i \in I$ is put into the jth bin for some $j \in \{1, \ldots, b\}$, then we do *not* process the job of client i on day j, otherwise we do. Since every item is put into exactly one bin, every client has a job that is scheduled to be processed on all except one day and since all bins are exactly filled, all scheduled jobs on every day can be processed before their deadline d. It follows that we have a $m - 1$-equitable set of schedules.

($\Leftarrow$): Assume we have a $m - 1$-equitable set of schedules. Then every client has a job on *exactly* all but one days that is processed. This follows from the fact that the total available processing time is *exactly* sufficient to process $m - 1$ jobs of every client. Let client i have a job that is *not* processed on day j. Then we put item i into the jth bin. Since the processing time $p_{i,j}$ is the same as the size of item i and the sum of the processing times of jobs that are *not* scheduled to be processed on the same day is exactly B, all bins are exactly filled. Hence, we have a valid distribution of items into bins. $\square$

Finally, we can observe that ESSD* is still weakly NP-hard for two days.

Observation 4 ESSD* *is NP-hard even if* $m = 2$.

Proof This can be shown with a straightforward reduction from Partition, where given a set $S = \{s_1, \ldots, s_{|S|}\}$ of integers, we are asked whether there is a partition of S into S_1 and S_2 (i.e., $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$) such that $\sum_{s \in S_1} s = \sum_{s \in S_2} s$. Partition is known to be weakly NP-hard [24].

Given an instance S of Partition we construct an instance of ESSD* with $n = |S|$ clients and $m = 2$ days in the following way. We set

$$p_{i,j} = s_i \text{ for all } i \in \{1, \ldots, |S|\} \text{ and } j \in \{1, 2\},$$

and we set $k = 1$ and $d = \frac{1}{2} \sum_{s \in S} s$ (note that in any yes-instance of Partition, the sum of all integers in S needs to be an even number). This can clearly be done in polynomial time.

($\Rightarrow$): Assume the Partition instance is a YES-instance. Then there is a partition of S into S_1 and S_2 such that $\sum_{s \in S_1} s = \sum_{s \in S_2} s = \frac{1}{2} \sum_{s \in S} s$. For each client i we schedule its job to be processed on day one if $s_i \in S_1$ and on day two otherwise. Since $\sum_{s \in S_1} s = \sum_{s \in S_2} s = \frac{1}{2} \sum_{s \in S} s$ we know that on both days the scheduled jobs finish before the deadline.

($\Leftarrow$): Assume we have a 1-equitable set of schedules. Then if client i has its job processed on day j, we put s_i into S_j. We first claim that this produces a partition of S. Clearly every $s_i \in S$ is at least in one of the sets S_1 and S_2, since otherwise client i has none of its jobs scheduled. Assume for contradiction that there is a client i that has its job scheduled on both days. Then the total processing time of both days is at least $s_i + \sum_{s \in S} s > \sum_{s \in S} s$, hence on one of the days not all jobs can finish before the deadline, a contradiction. It follows that $\sum_{s \in S_1} s \leq \frac{1}{2} \sum_{s \in S} s$ and $\sum_{s \in S_2} s \leq \frac{1}{2} \sum_{s \in S} s$ which implies that $\sum_{s \in S_1} s = \sum_{s \in S_2} s = \frac{1}{2} \sum_{s \in S} s$. $\square$

3.2 Algorithmic Results

We first show that we can solve ESSD in pseudo-polynomial time if the number m of days is constant. Note that this implies that ESSD is in XP when parameterized by the number of days if all processing times and the deadline are encoded in unary. Theorem 3 shows that we presumably cannot expect to be able to obtain an FPT algorithm for this case.

Theorem 5 ESSD *can be solved in* $O(d_{\max}^m \cdot \binom{m}{k} \cdot n)$ *time, where* $d_{\max} = \max_j d_j$.

Proof We give a dynamic programming algorithm for this problem. The table T maps from $\{0, \ldots, n\} \times \{1, \ldots, d_{\max}\}^m$ to $\{\texttt{true}, \texttt{false}\}$ and intuitively an entry $T[i, b_1, \ldots, b_m]$ is true if and only if it is possible to schedule k jobs of each client $\{1, \ldots, i\}$ such that the total processing time on day j is at most b_j for all days $j \in \{1, \ldots, m\}$. Formally, the table is defined as follows.

$$T[0, b_1, \ldots, b_m] = \texttt{true}$$
$$T[i, b_1, \ldots, b_m] =$$
$$\bigvee_{\{x_1, \ldots, x_k\} \in \binom{\{1, \ldots, m\}}{k}} T[i-1, \ldots, b_{x_1} - p_{i,x_1}, \ldots, b_{x_k} - p_{i,x_k}, \ldots]$$

Intuitively, we "guess" on which days we want to schedule jobs of client i and then look up whether there exists a set of k-equitable schedules for clients $\{1, \ldots, i-1\}$ to which we can add k jobs of client i such that the total processing time comply with the upper bounds.

It is easy to check that the input instance is a YES-instance if and only if $T[n, d_1, \ldots, d_m] = \texttt{true}$. The size of the table T is in $O(d_{\max}^m \cdot n)$ and computing one table entry takes $O(\binom{m}{k})$ time. Hence, we arrive at the claimed running time. $\square$

Next, we show that ESSD can be solved in polynomial time if the number n of clients is constant. In other words, we show that ESSD is in XP when parameterized by the number of clients.

Theorem 6 ESSD *can be solved in* $O((2k+2)^n \cdot m)$ *time.*

Proof We give dynamic programming algorithm for this problem. The table T maps from $\{1, \ldots, m\} \times \{0, \ldots, k\}^n$ to $\{\texttt{true}, \texttt{false}\}$. Entry $T[j, \ell_1, \ldots, \ell_n]$ is true if and only if it is possible to provide schedules for days $\{1, \ldots, j\}$ such that for each client i we have that its job is processed on exactly ℓ_i days ($\ell_i \leq k$). We have that

$$T[j, 0, \ldots, 0] = \texttt{true} \text{ for all } j \in \{1, \ldots, n\}, \text{ and}$$
$$T[j, \ell_1, \ldots, \ell_n] = \texttt{true}$$

if there exists a subset of clients $I^* \subseteq \{1, \ldots, n\}$ such that

$$\sum_{i \in I^*} p_{i,j} \leq d_j$$

and that

$$T[j-1, \ell_1 - \mathbb{1}_{1 \in I^*}, \ldots, \ell_n - \mathbb{1}_{n \in I^*}] = \texttt{true}$$

where $\mathbb{1}_{i \in I^*}$ is an indicator variable for the event that $i \in I^*$, i.e.,

$$\mathbb{1}_{i \in I^*} = \begin{cases} 1 & \text{if } i \in I^* \\ 0 & \text{otherwise} \end{cases}$$

It is easy to check that the input instance is a YES-instance if and only if $T[m, k, \ldots, k] = \texttt{true}$. The theorem follows from the fact that we have $O(m \cdot (k+1)^n)$ many $T[j, \ell_1, \ldots, \ell_n]$ values to compute and $O(2^n)$ possible I^* subsets to check for calculating any $T[j, \ell_1, \ldots, \ell_n]$ value. $\square$

We now strengthen the above result by showing that ESSD is in FPT when parameterized by n. To do this, we give an integer linear programm formulation for the problem and use a famous result by Lenstra Jr [44]. Note, however, that Theorem 6 is a purely combinatorial result and that the implicit running time of Theorem 7 is at least double exponential.

Theorem 7 ESSD *is in FPT when parameterized by the number of clients* n.

Proof First we partition the days into equivalence classes. We say that two days j and j' are equivalent if for any subset S of clients all jobs of S can be scheduled together on day j if and only if they can be scheduled together on day j'. Let $\mathcal{E}$ be the set of equivalence classes. Clearly, $|\mathcal{E}| \leq 2^{2^n}$. We write that $S \succ E$ for a set S of clients and an equivalence class E if the sum of the processing times of all jobs from S exceeds the deadline on every day from E.

We design an ILP with one variable $x_{E,S}$ for each pair of equivalence class $E \in \mathcal{E}$ and subset of clients S from E as follows.

$$x_{E,S} = 0 \qquad \text{if } S \succ E$$
$$\sum_{S:i \in S} \sum_{E \in \mathcal{E}} x_{E,S} \geq k \qquad \forall i \in \{1, \ldots, n\}$$
$$\sum_{S \subseteq \{1, \ldots, n\}} x_{E,S} = |E| \qquad \forall E \in \mathcal{E}$$

Since the number of variables is at most $2^n \cdot 2^{2^n}$, it follows by Lenstra Jr [44] that the ILP can be solved in FPT-time parameterized by n.

Given a solution to the ILP, we get a k-equitable schedule by scheduling for each variable $x_{E,S}$ the jobs of S on exactly $x_{E,S}$ days of the equivalence class E. By the third condition, this results in one schedule for every day. By the first condition none of the scheduled jobs is tardy. By the second condition, the schedule is k-equitable.

Vice versa, given a k-equitable schedule, we construct a feasible solution to the ILP by setting $x_{E,S}$ to be the number of days from equivalence class E scheduling exactly the jobs from S before the deadline. The first condition is then fulfilled by the definition of $S \succ E$. The second condition is fulfilled as the schedule is k-equitable. The third condition is fulfilled as there is exactly one schedule for each day. $\square$

In the remainder of this subsection, we investigate the canonical optimization version of ESSD* where we want to maximize k. Note that the existence of a polynomial-time approximation algorithm with any factor (i.e., an algorithm computing a solution for an instance $\mathcal{I}$ of value $\text{ALG}(\mathcal{I})$ such that $f(\mathcal{I}) \cdot \text{ALG}(\mathcal{I}) \geq \text{OPT}(\mathcal{I})$ for some function f) implies $P = NP$, since distinguishing between the cases $k = 0$ and $k = 1$ is NP-hard (see Theorem 3).

However, we will show that for any instance with optimal solution value $3k$, we can find a solution of value $2k$. We make a case distinction on k: we first show an algorithm for that case that $k \leq \frac{m}{2}$ and afterwards an algorithm for the case of $k > \frac{m}{2}$.

Lemma 2 *Given a YES-instance* $\mathcal{I} = (\{p_1, \ldots, p_n\}, m, d, k)$ *with* $k \leq \frac{m}{2}$ *of* ESSD*, *one can compute a solution to the instance* $\mathcal{I}' := (\{p_1, \ldots, p_n\}, m, d, k')$ *with* $k' := 2\lfloor \frac{k}{3} \rfloor$ *in* $O(n \cdot (k + \log n))$ *time.*

Proof We apply an algorithm similar to the so-called "First-Fit-Decreasing" algorithm for BIN PACK-ING [37]. Set $k' := 2\lfloor \frac{k}{3} \rfloor$. The algorithm works in the following steps.

1. Order the clients decreasingly by the processing time of their jobs.

2. Iterate through the clients in the computed order. For each client we schedule k' of their jobs on the first k' days that have enough space, i.e., after the jobs are scheduled the sum of processing times of the scheduled jobs for each day is at most d.
   Note that so far (i.e., without Step 3), the jobs of each client are scheduled in a block of k' consecutive days that starts a some day j with $j \bmod k' = 0$.

3. If there is a client i who cannot have k' of its jobs scheduled that way, then do the following:
   Note that when this happens for the first time, it means that all blocks of k' consecutive days that start at some day j with $j \bmod k' = 0$ are "full". We now make a case distinction on the number $m \bmod k'$ of days that are not part of any of these blocks.
   - *If Step 3 is invoked for the first time, then let i' be the client with smallest processing time scheduled on day $\lfloor \frac{2m}{3} \rfloor + 1$. Let j be the first day that has a job of client i' scheduled. Schedule jobs of clients i and i' to days $\{m - (m \bmod k') + 1, \ldots, m - (m \bmod k') + \frac{k'}{2}\}$ and replace the jobs of client i' that are scheduled on days $\{j, \ldots, j + \frac{k'}{2} - 1\}$ with jobs of client i.*
     If Step 3 is invoked for the second time, then output FAIL.
   - *If $m \bmod k' < \frac{k'}{2}$, then output FAIL.*

4. If all clients are processed, then output the schedules.

We first show that if the presented algorithm outputs a set of schedules, then the set is k'-equitable. If $m \bmod k' < \frac{k'}{2}$, then this is obvious. If $m \bmod k' \geq \frac{k'}{2}$, then we have to check that Step 3 of the algorithm does not produce infeasible schedules. Observe that in Step 3, we have that $p_{i'} \geq p_i$ since the clients are ordered by the processing time of their jobs and client i' is processed before client i. This means that replacing a jobs of client i' by a job of client i on some day cannot violate the deadline unless it was already violated before swapping the jobs. Observe that if $\mathcal{I}$ is a YES-instance, then there can be at most $\lfloor \frac{m}{k} \rfloor$ jobs with processing time more than $\frac{d}{2}$. Thus there are at most $\lfloor \frac{2m}{3} \rfloor$ days on which our algorithm schedules a job with processing time more than $\frac{d}{2}$. Since the algorithm processes the jobs in decreasing order, all jobs with length more than $\frac{d}{2}$ are scheduled only on the first $\lfloor \frac{2m}{3} \rfloor$ days. It follows that $p_{i'} \leq \frac{d}{2}$ since it is scheduled on day $\lfloor \frac{2m}{3} \rfloor + 1$. It follows that $p_i \leq p_{i'} \leq \frac{d}{2}$, and thus, the deadline is not violated on days $m - (m \bmod k') + 1, \ldots, m - (m \bmod k') + \frac{k'}{2}$. This implies that Step 3 always produces k'-equitable sets of schedules.

In the remainder of the proof we show that if the presented algorithm outputs FAIL, then $\mathcal{I}$ is a NO-instance. On an intuitive level, the main idea is to show that the first $\lceil \frac{2m}{3} \rceil$ days are "full" and the remaining $\lfloor \frac{m}{3} \rfloor$ days have at least $\lceil \frac{2m}{3} \rceil - k'$ jobs (in total) scheduled. This then allows us to show that the total processing time if k jobs of each client were scheduled exceeds $m \cdot d$, which implies that $\mathcal{I}$ is a NO-instance.

Since all jobs with length more than $\frac{d}{2}$ are scheduled only on the first $\lfloor \frac{2m}{3} \rfloor$ days, it follows that if the algorithm outputs FAIL, then the last $\lceil \frac{m}{3} \rceil$ days have either at least two jobs scheduled or none.

Assume that our algorithm outputs FAIL and let client i^* be the client that was processed when the algorithm output FAIL. Note that there are strictly less than $\frac{k'}{2}$ days with no jobs scheduled, independent on whether $m \bmod k' \leq \frac{k'}{2}$. Thus, among the last $\lfloor \frac{m}{3} \rfloor$ days, (strictly) less than $\frac{k'}{2}$ days have no jobs scheduled and all others have at least two jobs scheduled. Together with k' jobs of client i^* which are not scheduled at all, we have at least $2(\lfloor \frac{m}{3} \rfloor - \frac{k'}{2} + 1) + k' \geq 2\lceil \frac{m}{3} \rceil$ jobs, all of which have a processing time of at least p_{i^*}. Let the set of these jobs be called J^*. Since the jobs of client i^* could not be scheduled in the first $\lceil \frac{2m}{3} \rceil$ days, we know that the total processing time of all jobs from one of the first $\lceil \frac{2m}{3} \rceil$ days plus p_{i^*} or the processing time of any job in J^* is larger than the deadline d. Intuitively, this allows us to "distribute" the processing times of the jobs in J^* to the first $\lceil \frac{2m}{3} \rceil$ days (note that $|J^*| \geq \lceil \frac{2m}{3} \rceil$) and derive the following estimate: $k' \sum_{i:p_i \geq p_{i^*}} p_i > \lceil \frac{2m}{3} \rceil \cdot d$. Substituting k' with k and summing over all clients, we get $k \sum_{i \in \{1, \ldots, n\}} p_i > m \cdot d$, which is a contradiction to the assumption that $\mathcal{I}$ is a YES-instance.

Since First-Fit-Decreasing can be implemented in $O(n^* \log n^*)$ [36], where n^* is the number of elements, Steps 1 and 2 can be performed in time $O(n \log n + n \cdot k)$ by running First-Fit-Decreasing on the instance with one element of size p_i for each client i and $\lfloor \frac{m}{k'} \rfloor$ bins of size d, and then cloning the solution k' times. Step 3 clearly runs in $O(k)$ while Step 4 runs in constant time. The running time of $O(n(k + \log n))$ follows. □

We now turn to the case $k > \frac{m}{2}$.

Lemma 3 *Given a YES-instance $\mathcal{I} = (\{p_1, \ldots, p_n\}, m, d, k)$ with $k > \frac{m}{2}$ of ESSD*, one can compute a solution to the instance $\mathcal{I}' := (\{p_1, \ldots, p_n\}, m, d, k')$ with $k' := \lfloor \frac{2k}{3} \rfloor$ in $O(n \cdot (k + \log n))$ time.*

Proof We classify the clients into two groups based on the processing time of their jobs: A client i is *large* if $p_i > \frac{d}{3}$ and *small* otherwise (i.e., if $p_i \leq \frac{d}{3}$). Set $k' := \lfloor \frac{2k}{3} \rfloor$. We start with some basic observations:

1. There are no two clients i_1 and i_2 with $p_{i_1} + p_{i_2} > d$. Since $k > \frac{m}{2}$, every solution to $\mathcal{I}$ must schedule jobs of clients i_1 and i_2 at least once to the same day by the pigeonhole principle, which is impossible if $p_{i_1} + p_{i_2} > d$.
2. There are at most three large clients. Assume for contradiction that there are four large clients. Then, since $k > \frac{m}{2}$, by the pigeonhole principle there is one day that has three jobs from three of the four large clients scheduled, which is impossible since the total processing time on that day would exceed d.
3. The total processing time of all jobs that need to be scheduled cannot exceed $m \cdot d$, i.e., $k \sum_{i \in \{1,\dots,n\}} p_i \le m \cdot d$.
   Note that this implies that if a k'-equitable set of schedules schedules on each day jobs with total processing time larger than $\frac{2d}{3}$, then $\mathcal{I}$ is a NO-instance, since then $k \sum_{i \in \{1,\dots,n\}} p_i \ge \frac{3}{2} k' \sum_{i \in \{1,\dots,n\}} p_i > \frac{3}{2} m \frac{2d}{3} = m \cdot d$.

From now on we assume that the first two observations hold, otherwise $\mathcal{I}$ is a NO-instance.

Intuitively, we will mostly try to use the third observation to show that our algorithm is correct: We greedily fill up all days with jobs until no job of a small client fits in any day. If this happens and we do not have a k'-equitable set of schedules, then by the third observation we can deduce that we were facing a NO-instance. However, in order to do this, we first have to deal with some special cases explicitely (which are handled in Steps 2 and 3 of the algorithm in the next paragraph). If the total processing time of the jobs of all small clients is very small (i.e., at most $\frac{d}{3}$) we can construct a k'-equitable set of schedules directly. We also need to treat some cases where the total processing time of the jobs of all small clients is at most $\frac{2d}{3}$ separately, hence then we can have the case that we cannot schedule a job of any small client on a certain day and still the total processing time on that day does not exceed $\frac{2d}{3}$, which prevents us from applying the third obervation. Formally, we sort once in all clients by the processing times of their jobs, and then we compute a set of schedules in the following way.

1. If the sum of processing times of all small clients is at most $\frac{d}{3}$ and there are at least two large clients, then we do the following.
   We schedule the jobs of the up to three large clients one after another in the following way. We pick the k days having the most free processing time and schedule a job of client whose job we currently schedule on these days. If these schedules exceed the deadline on one day, then we output FAIL.

Now we pick $\lceil \frac{k}{3} \rceil$ days where the first large client has a job scheduled, we remove that job and replace it with jobs of all small clients. Next, we pick $\lceil \frac{k}{3} \rceil$ *different* days where the second large client has a job scheduled, we remove that job and replace it with jobs of all small clients.

2. If the sum of processing times of all small clients is at most $\frac{2d}{3}$ and there are at most two large jobs, then we do the following.
   - If there are no large clients, then we schedule all jobs of all small clients on the first k' days.
   - If there is only one large client, then we schedule the job of the large client on the first k' days and on the $m - k'$ remaining days we schedule jobs of all small clients.
     If $m < 2k'$, then we recursively find a $\lfloor \frac{2}{3}(k + k' - m) \rfloor$-equitable schedule for the small clients on the first k days where the deadline is set to $d - p_\ell$, where p_ℓ is the processing time of the job of the large client.
   - If there are two large clients and $k' < \frac{m}{2}$, then we schedule jobs of the two large clients on the first k' days and jobs of all small clients on the last k' days.
3. We schedule the jobs of the up to three large clients one after another in the following way. We pick the k' days having the most free processing time and schedule a job of client whose job we currently schedule on these days. If these schedules exceed the deadline on one day, then we output FAIL.
4. We schedule the jobs of the small clients of after another in the following fashion. We fix an order of the small clients and create a list L repeating this order k' times. We process the days from the first one to the last one as follows. Until the list L gets empty, we schedule the job of the first client i in L and delete (this appearance of) i from L, unless the job of i is already scheduled on this day, or the processing time of this job together with the processing time of all jobs already scheduled on this day exceeds the deadline. If the list L is non-empty after we processed the last day, then we return FAIL.

Assuming the algorithm does not recurse in Step 2, it is easy to check that if the algorithm does not output FAIL, then we found a k'-equitable set of schedules.

If the algorithm recurses in Step 2, then the large job is scheduled k' times and every small job is scheduled $m - k' + \lfloor \frac{2}{3}(k + k' - m) \rfloor$ times. Note that using $k' \le \min\{m, \frac{2k}{3}\}$, we get $m - k' + \frac{2}{3}(k + k' - m) = \frac{m}{3} + \frac{2k}{3} - \frac{k'}{3} \ge \frac{k'}{3} + k' - \frac{k'}{3} = k'$. Thus, due to the integrality of m and k', every small job is scheduled at least k' times.

In the remainder of the proof we show that if the algorithm outputs FAIL, then $\mathcal{I}$ is a NO-instance.

Assume that the algorithm outputs FAIL in Step 1. Since we assume that the first basic observation holds this can only happen if there are three large clients and the algorithm schedules one job of each large client to one day and all other days have two jobs of large clients scheduled. This can only happen if $3k > 2m$, however then, by the pigeonhole principle, any feasible solution would have to schedule jobs of each of the three large clients on the same day. This is a contradiction to the assumption that $\mathcal{I}$ is a YES-instance.

By the same argument, we have that if the algorithm outputs FAIL in Step 3, then $\mathcal{I}$ is a NO-instance.

Now assume that the algorithm outputs FAIL in Step 4. Since Step 4 was applied, the sum of processing of all small clients is larger than $\frac{2d}{3}$, or the processing time of all small clients exceeds $\frac{d}{3}$ and on each day, at least one large job is scheduled. This implies that for each day the sum of processing times of jobs scheduled at that day is larger than $\frac{2d}{3}$, since otherwise the algorithm would have scheduled the job from the next client in L. However then, by the third basic observation, we know that $\mathcal{I}$ is a NO-instance.

Finally, in Step 3, the algorithm may output FAIL only in the recursive call. Fix a solution to $\mathcal{I}$ which schedules the large jobs on the first k days. Thus, on the first k' days, this solution has to schedule each of the small jobs at least $m-k'$ times, and has only $d-p_\ell$ time on each of these days, where p_ℓ is the processing time of the large job. It follows that the created instance admits a $k - (m - k')$-equitable-schedule. Thus, by induction, our algorithm finds a $k + k' - m$-equitable-schedule on this instance and does not output FAIL.

Except for the recursion, all of Steps 1-4 can clearly be performed in $O(n \cdot k)$. Since we sort the clients by the processing time of their job, calling the recursion in Step 2 can be done in constant time, as only the large jobs (which are first up to three jobs) need to be removed from the instance and k' and d need to be adjusted. Thus, a total running time of $O(n(k + \log n))$ follows. □

Now we can combine Lemma 2 and Lemma 3 to get the following result.

Theorem 8 *Given a YES-instance $\mathcal{I} = (\{p_1, \ldots, p_n\}, m, d, k)$ of ESSD*, one can compute a solution to an instance $\mathcal{I}' := (\{p_1, \ldots, p_n\}, m, d, k')$ with $\lfloor 2\frac{k}{3} \rfloor \geq k' \geq 2\lfloor \frac{k}{3} \rfloor$ in $O(n \cdot (k + \log n))$ time.*

We leave as an open question whether a similar result can be obtained for ESSD.

4 Precedence Constraints

In this section, we investigate the computational complexity of EQUITABLE SCHEDULING WITH PRECEDENCE CONSTRAINTS.

4.1 Hardness Results

The hardness result from Theorem 3 and Corollary 1 for ESSD* can easily be adapted to ESPC* by modeling processing times by paths of appropriate length in the precedence DAG. Hence, we get the following result.

Corollary 2 *ESPC* is NP-hard and W[1]-hard when parameterized by the number of days m even if $k = 1$ or $m - k = 1$, and the precedence DAG consists of disjoint paths.*

Proof (sketch) We use the same idea as in the proof of Theorem 3 and Corollary 1. For each client i the reduction in the proof of Theorem 3 created one job of each day j with processing time p_i that can be encoded in unary. This allows us to introduce $p_i - 1$ additional dummy clients for each client i. In the precedence DAG for each day j we add a directed path of length p_i where the job of client i is the last one in the path and the other jobs in the path are the ones of the dummy clients for client i in some arbitrary order. This means that of the job of client i to be scheduled on a day j, the jobs of the dummy clients for client i also have to be scheduled on that day which simulated the processing time p_i. □

For the setting where we do not have that all days look the same, we get NP-hardness even for two days and common deadlines for all jobs in each of the days.

Theorem 9 *ESPC is NP-hard even if $k = 1$ and $m = 2$.*

Proof We present a polynomial-time many-one reduction from CLIQUE, where given a graph $H = (U, F)$ and an integer h, we are asked to decide whether H contains a complete subgraph with h vertices. This problem is known to be NP-complete [38]. Given a graph $H = (U, F)$ and an integer h we construct an instance of ESPC as follows. Assume that the vertices in U are ordered in some fixed but arbitrary way, that is, $U = \{v_1, v_2, \ldots, v_{|U|}\}$.

- For each vertex $v \in U$ we create one "vertex client" i_v and for each edge $e \in F$ we create one "edge client" j_e.
- For day one we create the precedence DAG G_1 where for all $\ell \in [|U| - 1]$ we have that $(i_{v_\ell}, i_{v_{\ell+1}}) \in E_1$ and for all $e \in F$ we have that $(i_{v_{|U|}}, j_e) \in E_1$.

That is, the precedence DAG is a directed path containing all jobs of vertex clients and all jobs of edge clients are out-neighbors of the job of the last vertex client in the path. Furthermore, we set the deadline d_1 for day one to $|U| + |F| - \binom{h}{2}$.

- For day two we create the precedence DAG G_2 where for all $v \in U$ we have that $(i_v, j_e) \in E_2$ if and only if $v \in e$. That is, for each edge e of H the precedence DAG contains two arcs from the jobs of the vertex clients corresponding to the endpoints of e to the job of the edge client corresponding to e. Furthermore, we set the deadline d_2 for day two to $h + \binom{h}{2}$.
- We set $k = 1$.

Clearly, the reduction can be performed in polynomial time. Intuitively, day one is a "selection gadget". The deadline and the precedence DAG are chosen in a way such that all jobs except the ones of $\binom{h}{2}$ edge clients can be scheduled. The second day is a "validation gadget" that ensures that the edges corresponding to the $\binom{h}{2}$ edge clients that have no job scheduled on day one form a clique in H.

($\Rightarrow$): Assume that H contains a clique $X \subseteq U$ of size h. On day one, we schedule all jobs of vertex clients i_v in the order prescribed by the precedence DAG G_1. Then for all edges $e \in F$ such that not both endpoints of e are in X we schedule the job of the corresponding edge client j_e. Note that G_1 allows us to do this, since all jobs of vertex clients are already scheduled. Furthermore, the deadline of day one allows us to schedule jobs of all but $\binom{h}{2}$ clients. Since the vertices of X form a clique, there are exactly $\binom{h}{2}$ edges that have both their endpoints in X. Hence all jobs that are scheduled on day one finish before the deadline.

On day two, we first schedule all jobs of vertex clients i_v with $v \in X$. Then we schedule the jobs of edge clients j_e with $e \subseteq X$, that is, both endpoints of e are part of the clique X. Note that G_2 allows us to schedule the jobs of these edge clients since we already scheduled the jobs of the vertex clients corresponding to the endpoints the edges corresponding to the jobs of these edge clients. Note that those edge clients are exactly the ones that do not have their jobs scheduled on day one. Furthermore, the total number of jobs scheduled on day two is $h + \binom{h}{2}$, hence they all finish before the deadline. It follows that we have found a set of 1-equitable schedules.

($\Leftarrow$): Assume that there is a set of 1-equitable schedules. Note that on day one, the precedence DAG required that the jobs of all vertex clients are scheduled first, and then an arbitrary set of $|F| - \binom{h}{2}$ jobs of edge clients can be scheduled. Let $F^\star \subseteq F$ be the set of edges such that the corresponding edge clients do not

have a job scheduled on day one. Note that $|F^\star| \geq \binom{h}{2}$ and that all edge clients corresponding to edges in $F^\star$ have their job scheduled on day two, otherwise the set of schedules would not be 1-equitable. The precedence DAG G_2 for day two requires that if a job of an edge client j_e is scheduled, the jobs of the vertex clients corresponding to the endpoints of e need to be scheduled before. The deadline of day two allows for at most h additional jobs to be scheduled, hence there need to be h jobs that can be scheduled on day two such that all precedence constraints for the jobs of edge clients corresponding to edges in $F^\star$ are fulfilled. Note that by construction of G_2 we can assume that all additionally scheduled jobs belong to vertex clients. Let $U^\star \subseteq U$ be the set of vertices corresponding to vertex clients that have a job scheduled on day two. We already argued that $|U^\star| \leq h$. However, we also have that $|U^\star| \geq h$ since otherwise, by the pidgeon hole principle, there is at least one edge client corresponding to an edge in $F^\star$ that does not have the precedence constraints of its job fulfilled. It follows that $|U^\star| = h$ which implies that the vertices in $U^\star$ form a clique in H. $\square$

In the following, we present some hardness results that show that even further restrictions on the precedence DAG presumably cannot yield polynomial-time solvability.

Theorem 10 ESPC *is NP-hard even if $k = 1$, $d = 3$, and the precedence DAG of each day consists of at most two disjoint paths.*

Proof We present a polynomial-time many-one reduction from the restriction of MONOTONE NOT-ALL-EQUAL-SAT, where every variable appears in exactly three clauses, every clause contains two or three variables, and every clause contains only non-negated literals. Given a set of clauses, we are asked whether there is an assignment of truth values to the variables such that every clause contains at least one variable that is set to true and at least one variable that is set to false. This problem is known to be NP-complete [18]. By Hall's Marriage Theorem [28], the incidence graph contains a matching M which leaves no variable unmatched. By considering the clause to which M matches to a given variable x as the last occurence of x, we may assume that the last occurrence of variable x is the last variable of the clause. Let a be the number of variables, b_2 be the number of clauses with two variables, and b_3 the number of clauses with three variables. We construct an instance of ESPC as follows.

- We set the deadline to three, i.e., $d = 3$, and we set $k = 1$.

- For each variable x_j, we create six clients: $i_1^{(j,T)}$, $i_2^{(j,T)}$, $i_3^{(j,T)}$, $i_1^{(j,F)}$, $i_2^{(j,F)}$, $i_3^{(j,F)}$.
- We create three "dummy clients" $i_1^{(D)}$, $i_2^{(D)}$, $i_3^{(D)}$.
- We create $m = 1 + a + b_2 + 2b_3$ days: one "dummy day", a variable days, and $b_2 + 2b_3$ clause days.

Dummy day: For the first day we create a precedence DAG that is one directed path starting with jobs of clients $i_1^{(D)}$, $i_2^{(D)}$, $i_3^{(D)}$ and then the jobs of all remaining clients in an arbitrary order.

Variable days: For variable x_j we create day $j + 1$ with a precedence DAG that consists of two directed paths. The first path contains jobs of clients $i_1^{(j,T)}$, $i_2^{(j,T)}$, $i_3^{(j,T)}$ in that order. The second path starts with jobs of clients $i_1^{(j,F)}$, $i_2^{(j,F)}$, $i_3^{(j,F)}$ in that order and then the jobs of all remaining clients in an arbitrary order.

Clause days: Let (x_{j_1}, x_{j_2}) be the jth clause containing two variables and let it contain the t_1th and t_2th appearence of x_{j_1} and x_{j_2}, respectively. We create day $a + j + 1$ with a precedence DAG containing of the two paths $i_1^{(D)}, i_{t_1}^{(j_1,T)}, i_{t_2}^{(j_2,F)}$ and $i_2^{(D)}, i_{t_1}^{(j_1,F)}, i_{t_2}^{(j_2,T)}$. Let $(x_{j_1}, x_{j_2}, x_{j_3})$ be the jth clause containing three variables and let it contain the t_1th, t_2th, and t_3th appearence of x_{j_1}, x_{j_2}, and x_{j_3}, respectively. We create days $a + b_2 + 2j$ and $a + b_2 + 2j + 1$ with precedence DAGs consisting of two directed paths. On day $a + b_2 + 2j$, the first path contains jobs of clients $i_{t_1}^{(j_1,F)}, i_{t_2}^{(j_2,F)}, i_{t_3}^{(j_3,T)}$ in that order. The second path contains jobs of clients $i_{t_1}^{(j_1,T)}, i_{t_2}^{(j_2,T)}, i_{t_3}^{(j_3,F)}$ in that order. On day $a + b_2 + 2j + 1$, the first path starts with jobs of clients $i_1^{(D)}, i_{t_1}^{(j_1,F)}, i_{t_2}^{(j_2,T)}$ in that order and then the jobs of all remaining clients in an arbitrary order. The second path contains jobs of clients $i_2^{(D)}, i_{t_1}^{(j_1,T)}, i_{t_2}^{(j_2,F)}$ in that order.

This finishes the construction. Since maximum matchings on bipartite graphs can be computed in polynomial time [34], the reduction runs in polynomial time.

($\Rightarrow$) Assume we have a satisfying assignment of the Monotone Not-All-Equal-SAT formula. We produce a set of k-equitable schedules for $k = 1$ as follows. On day 1, we schedule the jobs of clients $i_1^{(D)}, i_2^{(D)}, i_3^{(D)}$. On each variable day, we schedule either jobs of clients $i_1^{(j,T)}, i_2^{(j,T)}, i_3^{(j,T)}$ or jobs of clients $i_1^{(j,F)}, i_2^{(j,F)}, i_3^{(j,F)}$. We do the latter if variable x_j is set to true and the former otherwise. For each clause containing exactly two variables x_{j_1} and x_{j_2}, we schedule jobs of clients $i_1^{(D)}, i_{t_1}^{(j_1,T)}, i_{t_2}^{(j_2,F)}$ if x_{j_1} is set to true, while we schedule jobs of clients $i_2^{(D)}, i_{t_1}^{(j_1,F)}, i_{t_2}^{(j_2,T)}$ otherwise. For each clause containing three variables x_{j_1}, x_{j_2}, and x_{j_3}, we schedule jobs of clients $i_{t_1}^{(j_1,F)}, i_{t_2}^{(j_2,F)}, i_{t_3}^{(j_3,T)}$ on the first clause day if x_{j_3} is set to true and jobs of clients $i_{t_1}^{(j_1,T)}, i_{t_2}^{(j_2,T)}, i_{t_3}^{(j_3,F)}$ otherwise. On

the second clause day, we schedule the jobs of clients $i_1^{(D)}, i_{t_1}^{(j_1,F)}, i_{t_2}^{(j_2,T)}$ if either x_{j_1} is set to false and x_{j_3} is set to false or x_{j_2} is set to true and x_{j_3} is set to true. Otherwise, we schedule the jobs of clients $i_2^{(D)}, i_{t_1}^{(j_1,T)}, i_{t_2}^{(j_2,F)}$.

It is easy to verify that all clients have at least on of their jobs scheduled.

($\Leftarrow$) Assume that we have a k-equitable set of schedules for $k = 1$. On day 1, we may assume that jobs of clients $i_1^{(d)}, i_2^{(d)}, i_3^{(d)}$ are scheduled, since it is never beneficial to leave slots empty. First we show that one each variable day either jobs of clients $i_1^{(j,T)}, i_2^{(j,T)}, i_3^{(j,T)}$ or jobs of clients $i_1^{(j,F)}, i_2^{(j,F)}, i_3^{(j,F)}$ are scheduled: Assume for a contradiction that this is not true for variable x_j. Since the third appearance of x_j is the last variable of the corresponding clause, either job of client $i_{t_3}^{(j_3,T)}$ or $i_3^{(j_3,F)}$ is not scheduled, a contradiction. We claim that setting variable x_j to true if and only if jobs of clients $i_1^{(j,F)}, i_2^{(j,F)}, i_3^{(j,F)}$ are scheduled on day $j + 1$ yields a satisfying assignment.

Consider the jth clause (x_{j_1}, x_{j_2}) with two variables, containing the t_1th and t_2th appearance of x_{j_1} and x_{j_2}, respectively. By the precedence constraints, jobs of clients $i_{t_1}^{(j_1,T)}, i_{t_1}^{(j_1,F)}, i_{t_2}^{(j_2,T)}, i_{t_2}^{(j_2,F)}$ can only be scheduled on day $j_1 + 1$, day $j_2 + 1$ or day $a + 1 + j$. If x_{j_1} and x_{j_2} are both set to true, then the jobs of both $i_{t_1}^{(j_1,T)}$ and $i_{t_2}^{(j_2,T)}$ need to be scheduled on day $a + 1 + j$, which is impossible. Similarly, if x_{j_1} and x_{j_2} are both set to false, then the jobs of both $i_{t_1}^{(j_1,F)}$ and $i_{t_2}^{(j_2,F)}$ need to be scheduled on day $a + 1 + j$, which is impossible.

Consider the jth clause $(x_{j_1}, x_{j_2}, x_{j_3})$ with three variables, containing the t_1th, t_2th, and t_3th appearance of x_{j_1}, x_{j_2}, and x_{j_3}, respectively. By the precedence constraints, jobs of clients $i_{t_1}^{(j_1,T)}, i_{t_1}^{(j_1,F)}, i_{t_2}^{(j_2,T)}, i_{t_2}^{(j_2,F)}, i_{t_3}^{(j_3,T)}, i_{t_3}^{(j_3,F)}$ can only be scheduled on day $j_1 + 1$, day $j_2 + 1$, day $j_3 + 1$, day $a + b_2 + 2j$ or day $a + b_2 + 2j + 1$. If x_{j_1}, x_{j_2}, and x_{j_3} are set to true, then the job of client $i_{t_3}^{(j_3,T)}$ needs to be scheduled on the first clause day. Thus, jobs of both $i_{t_1}^{(j_1,T)}$ and $i_{t_2}^{(j_2,T)}$ need to be scheduled on the second clause day, which is impossible. The case that x_{j_1}, x_{j_2}, and x_{j_3} are set to false leads to a contradiction by symmetric arguments. $\square$

We remark that by introducing additional dummy clients, the reduction for Theorem 10 can be modified in a way that the precedence DAGs consists of disjoint paths of constant length. Hence, we get the following result.

Corollary 3 ESPC *is NP-hard even if $k = 1$ and the precedence DAG of each day consists of disjoint paths of length at most four.*

Proof (sketch) In the proof of Theorem 10 we have that the precedence DAG consists of two disjoint paths on each day. However,s the deadline for each day is $d = 3$, which means that every job that is located in one of the paths at a distance larger than three from the source of the path cannot be scheduled on that day. We can achieve the same by introducing at most $3 \cdot n$ dummy clients and n additional days (where n is the number of clients in the instance constructed in the proof of Theorem 10). On the new days we make sure that all jobs of the dummy clients can be scheduled. That can be done by using a precedence DAG that consist of disjoint paths of length four where the first three jobs of each path stem from dummy clients and the last job of each path (which cannot be scheduled before the deadline) stems from an original client. For each of the original days we modify the precedence DAG in the following way. We replace each of the disjoint paths by its first four vertices. For each job that was located at a distance larger than four from the source of the path, we introduce a new path of length four to the precedence DAG that starts with three jobs of dummy clients and then the job from the original client. This ensures that this job now also cannot be scheduled on that day. Note that we have introduced sufficiently many dummy clients to be able to do this. The jobs of the (potentially) remaining dummy clients are distributed arbitrarily into disjoint paths of length at most four. Note that now all precedence DAGs consist of disjoint paths of length at most four. □

4.2 Algorithmic Result

In the following, we give an ILP formulation for ESPC to obtain fixed-parameter tractability for the number of clients that are incident to an arc in at least one precedence DAG.

Theorem 11 ESPC *is fixed-parameter tractable when parameterized by the number of clients that are incident to an arc in at least on precedence DAG.*

Proof Let $\mathcal{I}$ be an instance of ESPC. We assume without loss of generality that $d_j \leq n$ for all days $j \in \{1, \ldots, m\}$ (since on every day at most n jobs can be scheduled, we can replace the deadline by n otherwise). Let A be the set of clients incident to at least one arc appearing in some precedence DAG. Let $\alpha := |A|$ and β be the number of arcs appearing in at least one incidence DAG. Note that $\frac{\alpha}{2} \leq \beta \leq \binom{\alpha}{2}$. Note that the number of different precedence DAGs is at most 2^β. For a precedence DAG G and a deadline $d \in \{1, \ldots, m\}$, let $\gamma(G, d)$ denote the number of days with precedence DAG G

and deadline d. We define $\gamma(G) := \sum_{r=1}^n \gamma(G, r)$ and $\gamma^{\leq d}(G) := \sum_{r=1}^d \gamma(G, r)$.

We construct an integer linear program (ILP) as follows. For each precedence DAG G, subset $A' \subseteq A$, and $d \in \{n - \alpha + 1, \ldots, n\}$, we add a variable $x_{G, A', d}$, indicating on how many days with precedence DAG G and deadline d exactly the jobs from clients in A' are scheduled. Additional, for each precedence DAG G and subset $A' \subseteq A$, we add a variable $x_{G, A'}^{\leq n - \alpha}$, indicating on how many days with precedence DAG G and deadline at most $n - \alpha$ the jobs from clients in A' are scheduled. Furthermore, there are the constraints specified by Equations 1, 2, 3, 4, 5, 6, 7, and 8.

The number of variables in this ILP is upperbounded by $\alpha 2^{\alpha + \beta}$ and therefore it can be solved in FPT-time with respect to $\alpha + \beta$ by Lenstra Jr [44]. We now show that any solution to this ILP corresponds to a feasible schedule, and each solution to $\mathcal{I}$ corresponds to a solution to the ILP.

($\Rightarrow$): Let x be a feasible solution to the ILP. For each precedence DAG G and each $d \in \{n - \alpha + 1, \ldots, n\}$, we schedule on $x_{G, A', d}$ of the days with precedence DAG G and deadline d the jobs of clients contained in A', and no job of a client from $A \setminus A'$. For each precedence DAG G, we schedule on $x_{G, A'}^{\leq n - \alpha}$ of the $\gamma^{\leq n - \alpha}(G)$ days with precedence DAG G and deadline at most $n - \alpha$ the jobs of clients contained in A', and no job of a client from $A \setminus A'$. We do this in such a way that for each $A', A'' \subseteq A$ with $|A'| < |A''|$, it holds that the deadline of the days on which A'' is scheduled is at least the deadline of the days on which A' is scheduled. We iterate from day 1 to m. Let $A_j \subseteq A$ be the set of clients which have already a job scheduled on day j. For each day j, as long as there are less than d_j jobs scheduled on this day, and there is one job from a client in $\{1, \ldots, n\} \setminus A$ which has not been scheduled on this day, we pick a client $i \in \{1, \ldots, n\} \setminus A$ whose job has been scheduled fewest time till now, and schedule its job on day j. Note that this procedure ensures that for two jobs $i, i' \in \{1, \ldots, n\} \setminus A$, during any point of the procedure, job i is scheduled at most once more than job i'. Thus, no job is scheduled twice on one day.

Equations 1 and 2 ensure that each precedence DAG is considered exactly the number of times it actually appears in $\mathcal{I}$. Equation 3 ensures that every client in A has at least k of its jobs scheduled, while Equation 8 ensures this for all clients in $\{1, \ldots, n\} \setminus A$: On a day j with $d_j \in \{n - \alpha + 1, \ldots, n\}$, we schedule $\min\{d_j - |A'|, n - |A|\}$ jobs of these clients, while on a day j with $d_j \leq n - \alpha$, we schedule $d_j - |A_j'|$ jobs of these clients. Since jobs of client i are scheduled at most once more than jobs of clients i', it follows that the jobs of every client are scheduled at least k times. Equations 6

$$\sum_{A' \subseteq A} x_{G,A',d} = \gamma(G,d) \qquad \forall d \in \{n - |A| + 1, \ldots, n\}, \text{ precedence DAGs } G \tag{1}$$

$$\sum_{A' \subseteq A} x_{G,A'}^{\leq n - \alpha} = \gamma^{\leq n - \alpha}(G) \qquad \forall \text{ precedence DAGs } G \tag{2}$$

$$\sum_{A':i \in A'} \sum_{G} \left(x_{G,A'}^{\leq n - \alpha} + \sum_{d=n-\alpha+1}^{n} x_{G,A',d} \right) \geq k \qquad \forall i \in A \tag{3}$$

$$x_{G,A',d} = 0 \qquad \text{if } |A'| > d \tag{4}$$

$$\sum_{A' \subseteq A : |A'| \leq d} x_{G,A'}^{\leq n - \alpha} \geq \gamma^{\leq d}(G) \qquad \forall d \in \{1, \ldots, n - \alpha\} \tag{5}$$

$$x_{G,A'} = 0 \qquad \forall A', G : \exists (i, i') \in E(G) \text{ with } i \notin A' \wedge i' \in A' \tag{6}$$

$$x_{G,A',d} = 0 \qquad \forall A', G, d \in \{n - \alpha + 1, \ldots, n\} : \exists (i, i') \in E(G)$$
$$\text{with } i \notin A' \wedge i' \in A' \tag{7}$$

$$\sum_{A' \subseteq A, G, d \in \{n-\alpha+1,\ldots,n\}} \min\{d - |A'|, n - \alpha\} \cdot x_{G,A',d} + \sum_{j \in \{1,\ldots,m\}: d_j \leq n - \alpha} d_j - \sum_{A' \subseteq A, G} |A'| \cdot x^{\leq n - \alpha}(G, A') \geq k(n - |A|) \tag{8}$$

and 7 ensure that the precedence DAG is obeyed on every day. Equation 4 ensures that for every day with deadline at least $n - \alpha + 1$, the jobs scheduled on this day can be performed before the deadline. For all days with deadline at most $n - \alpha$, this is ensured by Equation 5: This inequality ensures that for any precedence graph G and any $d \leq n - \alpha$, there are at least $\gamma^{\leq d}(G)$ many days on which at most d jobs from A are scheduled, and consequently, we schedule at most d_j jobs on each such day j.

($\Leftarrow$): Consider a feasible schedule. We set $x_{G,A',d}$ to be the number of days with precedence DAG G and deadline $d \in \{n - \alpha + 1, \ldots, n\}$ on which all jobs from clients in A' but no job from a client in $A \setminus A'$ is scheduled. Thus, Equation 1 is fulfilled. Similarly, let $x_{G,A'}^{\leq n - \alpha}$ be the number of days with precedence DAG G and deadline at most $n - \alpha$ on which all jobs from clients in A' but no job from a client in $A \setminus A'$ is scheduled. It follows that Equation 2 is fulfilled. Since every client in A has at least k of its jobs scheduled, also Equation 3 is fulfilled. Since for any day j with deadline $d_j \in \{n - \alpha + 1, \ldots, n\}$ there are at most d_j jobs (from A) scheduled on this day, it follows that Equation 4 is fulfilled. Similarly, for each $d \in \{1, \ldots, n - \alpha\}$, on each of the days with deadline d and precedence graph G, the set A' of clients from A whose jobs are scheduled on this day fulfills $|A'| \leq d$, and therefore, Equation 5 holds. By the precedence constraints also Equations 6 and 7 are fulfilled. For any day $j \in \{1, \ldots, m\}$, let $|A'_j|$ be the set of clients from A whose jobs have been scheduled on day j. There are at most $\min\{d_j - |A'_j|, n - |A|\}$ jobs from $\{1, \ldots, n\} \setminus A'$ scheduled on day j. Since $\min\{d_j - |A'_j|, n - \alpha\} = d_j - |A'_j|$ if $d_j \leq n - \alpha$, and every client in $\{1, \ldots, n\} \setminus A$ has at least k of its jobs scheduled, it follows that Equation 8 is fulfilled. $\qquad \square$

We remark that this algorithm can easily be extended to the settings where jobs have equal processing time (which is not necessarily of unit length).

5 Conclusion

We have introduced a promising new framework for single machine scheduling problems. We investigated three basic single machine scheduling problems in this framework and we believe that it might also be interesting in other scheduling contexts.

We leave several questions open for future research. We believe that it would be promising to implement our approximation algorithm for ESSD* and, once provided with appropriate real-world data, test how well it performs in practice. The question whether we can get similar approximation results also for ESSD and ESPC remains unresolved. For ESPC, it is also remains open whether we can get similar combinatorial algorithms as for ESSD.

References

1. M. Adamu and A. Adewumi. Survey of single machine scheduling to minimize weighted number of tardy jobs. *Journal of Industrial and Management Optimization*, 10:219, 2014. 1
2. U. Bagchi, R. Sullivan, and Y. Chang. Minimizing mean squared deviation of completion times about a common due date. *Management Science*, 33(7): 894–906, 1987. 2
3. N. Bansal, T. Kimbrel, and M. Sviridenko. Job shop scheduling with unit processing times. *Mathematics of Operations Research*, 31(2):381–389, 2006. 2

4. P. Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal of Scheduling*, 2(6):245–252, 1999. 2

5. P. Baptiste. Scheduling equal-length jobs on identical parallel machines. *Discrete Applied Mathematics*, 103:21–32, 2000. 2

6. P. Baptiste, P. Brucker, S. Knust, and V. G. Timkovsky. Ten notes on equal-processing-time scheduling. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 2(2):111–127, 2004. 1

7. S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6): 600–625, 1996. 2

8. M. Bentert, R. van Bevern, and R. Niedermeier. Inductive k-independent graphs and c-colorable subgraphs in scheduling: a review. *Journal of Scheduling*, 22(1):3–20, 2019. 3

9. M. Bentert, R. Bredereck, P. Györgyi, A. Kaczmarczyk, and R. Niedermeier. A multivariate complexity analysis of the material consumption problem. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence, AAAI 2021*, pages 11755–11763. AAAI Press, 2021. 3

10. D. Bertsimas, V. F. Farias, and N. Trichakis. The price of fairness. *Operations Research*, 59(1):17–31, 2011. 2

11. K. Bülbül, S. Kedad-Sidhoum, and H. Şen. Single-machine common due date total earliness/tardiness scheduling with machine unavailability. *Journal of Scheduling*, 22:543–565, 2019. 2

12. R. Bredereck, A. Kaczmarczyk, and R. Niedermeier. Envy-free allocations respecting social networks. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2018*, pages 283–291, 2018. 2

13. P. Brucker, J. Hurink, and S. Knust. A polynomial algorithm for $p|p_j = 1, r_j, \text{outtree}| \sum c_j$ problem. *Mathematical Methods of Operations Research*, 56: 407–412, 2002. 2

14. C. Chen and R. Bulfin. Scheduling unit processing time jobs on a single machine with multiple criteria. *Computers & Operations Research*, 17(1):1–7, 1990. 2

15. H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3):181–194, 1990. 2

16. J. R. Correa and A. S. Schulz. Single-machine scheduling with precedence constraints. *Mathematics of Operations Research*, 30(4):1005–1021, 2005. 2

17. M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015. 3

18. A. Dehghan, M. Sadeghi, and A. Ahadi. On the complexity of deciding whether the regular number is at most two. *Graphs and Combinatorics*, 31(5): 1359–1365, 2015. 12

19. U. Dorndorf, E. Pesch, and T. Phan-Huy. A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. *Management Science*, 46 (10):1365–1384, 2000. 2

20. R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013. 3

21. J. Flum and M. Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer, 2006. 3

22. T. Fluschnik, P. Skowron, M. Triphaus, and K. Wilker. Fair knapsack. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI 2019*, pages 1941–1948. AAAI Press, 2019. 2

23. R. Ganian, T. Hamm, and G. Mescoff. The complexity landscape of resource-constrained scheduling. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1741–1747. ijcai.org, 2020. 3

24. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 7

25. M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete problems. *Theoretical Computer Science*, 1(3):237–267, 1976. 6

26. R. Graham, E. Lawler, J. Lenstra, and A. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 3:287–326, 1979. 1

27. S. Gupta, A. Jalan, G. Ranade, H. Yang, and S. Zhuang. Too many fairness metrics: Is there a solution? *SSRN*, 2020. URL https://dx.doi.org/10.2139/ssrn.3554829. 2

28. P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10:26–30, 1935. 12

29. K. Heeger, D. Hermelin, G. B. Mertzios, H. Molter, R. Niedermeier, and D. Shabtay. Equitable scheduling on a single machine. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence, AAAI '21*, pages 11818–11825. AAAI Press, 2021. 1

30. D. Hermelin, M. Pinedo, D. Shabtay, and N. Talmon. On the parameterized tractability of single

machine scheduling with rejection. *European Journal of Operational Research*, 273(1):67–73, 2019. 3

31. D. Hermelin, D. Shabtay, and N. Talmon. On the parameterized tractability of the just-in-time flow-shop scheduling problem. *Journal of Scheduling*, 22 (6):663–676, 2019. 3

32. D. Hermelin, G. Manoussakis, M. Pinedo, D. Shabtay, and L. Yedidsion. Parameterized multi-scenario single-machine scheduling problems. *Algorithmica*, 82(9):2644–2667, 2020. 3

33. D. J. Hoitomt, P. B. Luh, E. Max, and K. R. Pattipati. Scheduling jobs with simple precedence constraints on parallel machines. *IEEE Control Systems Magazine*, 10(2):34–40, 1990. 2

34. J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. 5, 13

35. K. Jansen, S. Kratsch, D. Marx, and I. Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1): 39–49, 2013. 6

36. D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3): 272–314, 1974. 9

37. D. S. Johnson, A. J. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974. 8

38. R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972. 11

39. A. Kumar and J. M. Kleinberg. Fairness measures for resource allocation. *SIAM Journal on Computing*, 36(3):657–680, 2006. 2

40. J. Lang and J. Rothe. Fair division of indivisible goods. In J. Rothe, editor, *Economics and Computation, An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division*, Springer Texts in Business and Economics, pages 493–550. Springer, 2016. 2

41. V. Lauff and F. Werner. Scheduling with common due date, earliness and tardiness penalties for multimachine problems: A survey. *Mathematical and Computer Modelling*, 40(5):637–655, 2004. 2

42. J. Lenstra and A. Rinnooy Kan. Complexity results for scheduling chains on a single machine. *European Journal of Operational Research*, 4(4):270 – 275, 1980. 1

43. J. K. Lenstra and A. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978. 2

44. H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983. 8, 14

45. W. L. Maxwell. On sequencing n jobs on one machine to minimize the number of late jobs. *Management Science*, 19(1):295–297, 1970. 1

46. M. Mnich and R. van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 100:254–261, 2018. 3

47. R. H. Möhring, M. Skutella, and F. Stork. Scheduling with and/or precedence constraints. *SIAM Journal on Computing*, 33(2):393–415, 2004. 2

48. J. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(2):102–109, 1968. 1

49. G. Mosheiov and S. Pruwer. On the minmax common-due-date problem: extensions to position-dependent processing times, job rejection, learning effect, uniform machines and flowshops. *Engineering Optimization*, 53(3):408–424, 2021. 2

50. G. Mosheiov and U. Yovel. Minimizing weighted earliness–tardiness and due-date cost with unit processing-time jobs. *European Journal of Operational Research*, 172(2):528–544, 2006. 2

51. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. 3

52. S. S. Panwalkar, M. L. Smith, and A. Seidmann. Common due date assignment to minimize total penalty for the one machine scheduling problem. *Operations Research*, 30(2):391–399, 1982. 2

53. D. Shabtay, K. Arviv, Y. Edan, and H. Stern. A combined robot selection and scheduling problem for flow-shops with no-wait restrictions. *Omega*, 43:96–107, 2014. 2

54. L. B. J. M. Sturm. A simple optimality proof of Moore's sequencing algorithm. *Management Science*, 17(1):116–118, 1970. 1

55. V. T'kindt, L. Shang, and F. D. Croce. Exponential time algorithms for just-in-time scheduling problems with common due date and symmetric weights. *Journal of Combinatorial Optimization*, 39:764–775, 2020. 2

56. T. Walsh. Fair division: The computer scientist's perspective. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4966–4972. ijcai.org, 2020. 2

57. T. Wang and O. Bellenguez-Morineau. The complexity of parallel machine scheduling of unit-processing-time jobs under level-order precedence constraints. *Journal of Scheduling*, 22:263–269, 2019. 2