Equitable Scheduling on a Single Machine

Klaus Heeger[*1], **Danny Hermelin**[2], **George B. Mertzios**[†3],
Hendrik Molter[‡1], **Rolf Niedermeier**[1], **Dvir Shabtay**[2]

[1] Faculty IV, Algorithmics and Computational Complexity, TU Berlin, Germany
[2] Ben Gurion University of the Negev, Beersheba, Israel
[3] Department of Computer Science, Durham University, UK
heeger@tu-berlin.de, hermelin@bgu.ac.il, george.mertzios@durham.ac.uk,
h.molter@tu-berlin.de, rolf.niedermeier@tu-berlin.de, dvirs@bgu.ac.il

Abstract

We introduce a natural but seemingly yet unstudied generalization of the problem of scheduling jobs on a single machine so as to minimize the number of tardy jobs. Our generalization 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.

Introduction

One of the most basic and fundamental scheduling problem 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. (1979). It is well-known that $1||\sum U_j$ is solvable in $O(n \log n)$ time (Maxwell 1970; Moore 1968; Sturm 1970), but gets e.g. NP-hard in case of simple (chain) precedence constraints even if all processing times p_j are the same (Lenstra and Rinnooy Kan 1980). There is also a more recent survey concerning the minimization of the weighted number of tardy jobs (Adamu and Adewumi 2014) and the problem has also been thoroughly studied for parallel machines (Baptiste et al. 2004).

Due to the ever increasing importance of high customer satisfaction, fairness-related issues are becoming more and more important in all areas of resource allocation (Bredereck, Kaczmarczyk, and Niedermeier 2018; Fluschnik et al. 2019; Lang and Rothe 2016; Walsh 2020) and particularly scheduling (Kumar and Kleinberg 2006).[1] For instance, in their seminal work Baruah et al. (1996) introduced the concept of proportionate progress, a fairness concept for resource allocation problems. They applied it to periodic scheduling by assigning resources to jobs according to their rational weights between 0 and 1, thereby aiming to make sure that a job never gets an entire slot (in the periodic schedule) ahead or behind. Nowadays, equity and fairness in resource allocation is a widely discussed topic, leading to considerations such as the "price of fairness" (Bertsimas, Farias, and Trichakis 2011) or to discussions about the abundance of fairness metrics (Gupta et al. 2020).

We study a very natural but seemingly novel extension of the $1||\sum U_j$ problem, taking into account a very basic aspect of equity among the customers in order to guarantee high customer satisfaction. Our task is to service n clients for m days, where each client has a single job to be scheduled for every day. This should be done in an equitable fashion. We focus on a very simple notion of equity where, given an integer parameter k, we request that each client receives satisfactory service in at least k out of the m days. In what follows, refer to k as *equity parameter*. It is important to note here that since all scheduling requests for all days and clients are assumed to be known in advance, we consequently still face an offline scenario of scheduling.

Consider the following motivating example. Imagine that a research group with n PhD students owns a single compute server, where each student has to submit a plan for their experiments for the next m days. Typically, at most one hour per experiment suffices and the needed computation time is known in advance. To make sufficient progress on their research, all students need to have regular access to the compute server for performing individual experiments. All students request access to the server every day, but due to high demand not all of them may be scheduled early enough during the day so that the experiments can still be evaluated the

---

---

[1] For instance, in 2018, ACM started its new conference series on "Fairness, Accountability, and Transparency (FAT)".

very same day. The chair of the group wants to guarantee a schedule for the compute server such that every student can evaluate their experiments on the same day in at least k out of m days. This is precisely the scenario we wish to model in the current manuscript.

Three Equitable Scheduling Variants. Our model can be formally described as follows: We wish to schedule the jobs of a set of n clients over 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 $|\{j \mid 1 \leq j \leq m \wedge C_{i,j} \leq d_{i,j}\}| \geq k$.

We consider three variants of EQUITABLE SCHEDULING, each corresponding to a well-studied variant of the $1||\sum U_j$ problem when restricted to a single day.

- 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 the EQUITABLE SCHEDULING WITH PRECEDENCE CONSTRAINTS (ESPC) problem, 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* 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 $\sigma_j(i_1) < \sigma_j(i_2)$.

For each of these variants, 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 $j_1, j_2 \in \{1, \ldots, m\}$.

Our Results. We are mainly interested in exact algorithms or algorithms with approximation guarantees. We study the (parameterized (Downey and Fellows 2013)) algorithmic complexity of all three main variants (and some further variations) discussed above. Our main findings are as follows.

- For ESUP we show that the problem can be solved in polynomial time by a reduction to the BIPARTITE MAXIMUM MATCHING problem. Our reduction can also be applied when jobs have release times, and when there is fixed number of machines available on each day.

- For ESSD and ESSD* we show strong NP-hardness and W[1]-hardness for the parameter number m of days even if $k = 1$. On the positive side, we show that ESSD can be solved in pseudo-polynomial time if the number m of days is constant and is in FPT for the parameter number n of clients. For ESSD* we give an algorithm that, for any k, computes a $2k$-equitable set of schedules if there exists a $3k$-equitable set of schedules.

- For ESPC and ESPC* we show NP-hardness and W[1]-hardness for the parameter number m of days even if $k = 1$ and the precedence DAG only consists of disjoint paths. For ESPC we also show NP-hardness for $k = 1$ if the precedence DAG either consists of a constant number of disjoint paths or disjoint paths of constant length. On the positive side, we can show that ESPC is in FPT for the parameter number n of clients.

Due to space constraints, some of our results (marked with $\star$) are (partially) deferred to the supplementary material.

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 release times, and when there is fixed number of machines available on each day.

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 $d_j^* = \max_{1 \leq i \leq n} d_{i,j}$ 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 $C > d_j^{(t)}$, then client i is unsatisfied on day t. 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.

- 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 $\sigma_j(i) = s_j + x_{i,j} + 1$.

Clearly 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 $\{\sigma_1, \ldots, \sigma_m\}$. $\square$

Observe that G has $O(mn)$ vertices and $O(mn^2 + m^2 n)$ edges, and it can be constructed in $O(mn^2 + m^2 n)$ time. Using the algorithm of Hopcroft and Karp (1973) for BIPARTITE MAXIMUM MATCHING, this gives us the following:

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

We remark that this algorithm is very flexible and can easily be extended to the setting where the jobs have release dates and where there are multiple parallel machines. The main idea is that the vertices in U represent time slots for jobs and a job has an edge to all time slots that are before

the job's deadline. To incorporate release dates we additionally remove edges to time slots that are "too early". Finally, to model parallel machines, we introduce copies of the vertices in U for each of the machines.

Single Deadline on Each Day

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

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

Theorem 3 ($\star$)**.** ESSD *is NP-hard even if $k = 1$ and $d = 3$.*

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.[2]

Theorem 4 ($\star$)**.** 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.*

Algorithmic Results. We first show that we can solve ESSD in pseudo-polynomial time if the number of days m 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 is encoded unary. Theorem 4 shows that we presumably cannot expect to be able to obtain an FPT algorithm for this case.

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

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

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

We remark that Theorems 5 and 6 are both achieved with dynamic programs.

We now strengthen Theorem 6 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 the famous result by Lenstra Jr (1983). 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 of clients S 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 (1983) 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 4).

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 8. *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 PACKING (Johnson et al. 1974). Set $k' := 2\lfloor \frac{k}{3} \rfloor$. The algorithm works in the following steps.

1. We sort all clients by the processing times of their jobs.

2. Iterate through the clients, starting with large processing times. 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 $j \mod k' = 0$.

3. If there is a client i who cannot have k' of its jobs scheduled that way, do the following:
   Note that when this happens for the first time, it means that all blocks of k' consecutive days that starts a some day j with $j \mod k' = 0$ are "full". We now make a case distinction on the number $m \mod k'$ 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 $\{m - (m \mod k') + 1, \ldots, m - (m \mod k') + \frac{k'}{2}\}$ 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 $m \mod k' < \frac{k'}{2}$, then output FAIL.

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

We first show that if the presented algorithm outputs a set of schedules, the set is k'-equitable. If $m \mod k' < \frac{k'}{2}$, then this is obvious. If $m \mod k' \geq \frac{k'}{2}$, 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 $m - (m \mod k') + 1, \ldots, m - (m \mod k') + \frac{k'}{2}$. 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 $m \mod k' \le \frac{k'}{2}$. 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 $2(\lfloor \frac{m}{3} \rfloor - \frac{k'}{2} + 1) + k' \ge 2\lceil \frac{m}{3} \rceil$ 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 $|J^*| \ge \lceil \frac{2m}{3} \rceil$) and derive the following estimate: $k' \sum_{i:p_i \ge p_{i^*}} p_i > \lceil \frac{2m}{3} \rceil \cdot d$. 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 $O(n^* \log n^*)$ (Johnson 1974), 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. $\qquad\square$

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

Lemma 9. *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 $p_i \le \frac{d}{3}$). Set $k' := \lfloor \frac{2k}{3} \rfloor$. We start with some basic obervations:

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 pidgeon-hole principle. This is infeasible 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 pidgeon-hole 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., $k \sum_{i \in \{1,\ldots,n\}} p_i \le m \cdot d$. 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 $k \sum_{i \in \{1,\ldots,n\}} p_i \ge \frac{3}{2} k' \sum_{i \in \{1,\ldots,n\}} p_i > \frac{3}{2} m \frac{2d}{3} = m \cdot d$.

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 explicitly (which are handled in Steps 1 and 2 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, 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, 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 $k' \leq \min\{m, \frac{2k}{3}\}$, we get $m-k'+\frac{2}{3}(k+k'-m) = \frac{m}{3}+\frac{2k}{3}-\frac{k'}{3} \geq \frac{k'}{3}+k'-\frac{k'}{3} = k'$. 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 pidgeon-hole 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 assumptoin 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.

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 Lemmas 8 and 9 to get the following result.

Theorem 10. *Given a YES-instance $\mathcal{I} = (\{p_1,\ldots,p_n\}, m, d, k)$ 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.*

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

Precedence Constraints

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

Hardness Results. The hardness result from Theorem 4 for ESSD* can easily be adapted to ESPC* by modelling processing times by paths of appropriate length in the precedence DAG. Hence, we get the following result (we omit the details here).

Corollary 11. *ESPC* is NP-hard and W[1]-hard when parameterized by the number of days m even if $k = 1$ and the precedence DAG consists of disjoint paths.*

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

Theorem 12 (⋆)**.** *ESPC is NP-hard even if $k = 1$, $d = 6$, and the precedence DAG of each day consists of three disjoint paths.*

Proof. We reduce from MONOTONE EXACTLY 1-IN-3-SAT, where every variable appears in exactly three clauses. This problem is known to be NP-complete (Porschen et al. 2014). Given a set of clauses, we are asked whether there is an assignment of truth values to the variables such that every clause contains exactly one variable that is set to true and two variables that are set to false. Let a be the number of variables and b be the number of clauses. We construct an instance of ESPC as follows.

- We set the deadline to six, i.e., $d = 6$, 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 9 "dummy clients" $i_1^{(d)}, \ldots, i_9^{(d)}$.
- We create $m = 2 + a + b$ days: two "dummy days", n variable days, and m clause days.

Dummy Days: For the first day we create a precedence DAG that is one directed path starting with jobs of clients $i_1^{(d)}, \ldots, i_6^{(d)}$ and then the jobs of all remaining clients in an arbitrary order. For the second day we create a precedence DAG that is one directed path starting with jobs of clients $i_4^{(d)}, \ldots, i_9^{(d)}$ and then the jobs of all remaining clients in an arbitrary order.

Variable Days: For variable x_j we create Day $j + 2$ with a precedence DAG that consists of two directed paths. The first path contains jobs of clients $i_1^{(d)}, i_2^{(d)}, i_3^{(d)}, i_1^{(j,T)}, i_2^{(j,T)}, i_3^{(j,T)}$ in that order. The second path starts with jobs of clients $i_4^{(d)}, i_5^{(d)}, i_6^{(d)}, i_1^{(j,F)}, i_2^{(j,F)}, i_3^{(j,F)}$ in that order and then the jobs of all remaining clients in an arbitrary order.

Clause Days: Let $(x_{j_1}, x_{j_2}, x_{j_3})$ be the jth clause and and let it contain the t_1th, t_2th, and t_3th appearance of x_{j_1}, x_{j_2}, and x_{j_3}, respectively. We we create Day $a + j + 2$ with a precedence DAG that consists of three directed paths. The first path contains

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

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

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

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

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

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

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

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

jobs of clients $i_1^{(d)}, i_2^{(d)}, i_3^{(d)}, i_{t_1}^{(j_1,T)}, i_{t_2}^{(j_2,F)}, i_{t_3}^{(j_3,F)}$ in that order. The second path contains jobs of clients $i_4^{(d)}, i_5^{(d)}, i_6^{(d)}, i_{t_1}^{(j_1,F)}, i_{t_2}^{(j_2,T)}, i_{t_3}^{(j_3,F)}$ in that order. The third path starts with jobs of clients $i_7^{(d)}, i_8^{(d)}, i_9^{(d)}, i_{t_1}^{(j_1,F)}, i_{t_2}^{(j_2,F)}, i_{t_3}^{(j_3,T)}$ in that order and then the jobs of all remaining clients in an arbitrary order.

This finishes the construction. It can clearly be done in polynomial time. The correctness deferred to the supplementary material. □

We remark that by introducing additional dummy clients, the reduction for Theorem 12 can be modified in a way that the precedence DAGs consists of disjoint paths of constant length. Hence, we get the following result (we omit the details here).

Corollary 13. ESPC *is NP-hard even if $k = 1$ and the precedence DAG of each day consists of disjoint paths of constant length.*

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 on precedence DAG.

Theorem 14 ($\star$). 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 $x_{\widetilde{G},A'}^{\leq n-\alpha}$, 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) to (8).

The number of variables in this ILP is bounded by $\alpha 2^{\alpha+\beta}$ and therefore can be solved in FPT-time with respect to $\alpha + \beta$ by Lenstra Jr (1983). The correctness proof for the ILP is deferred to the supplementary material. □

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* (Theorem 10) 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 and whether ESPC is in XP for the parameter number of days.

References

Adamu, M.; and Adewumi, A. 2014. Survey of single machine scheduling to minimize weighted number of tardy jobs. *Journal of Industrial and Management Optimization* 10: 219.

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

Baruah, S. K.; Cohen, N. K.; Plaxton, C. G.; and Varvel, D. A. 1996. Proportionate Progress: A Notion of Fairness in Resource Allocation. *Algorithmica* 15(6): 600–625.

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

Bertsimas, D.; Farias, V. F.; and Trichakis, N. 2011. The Price of Fairness. *Operations Research* 59(1): 17–31.

Bredereck, R.; Kaczmarczyk, A.; and Niedermeier, R. 2018. Envy-Free Allocations Respecting Social Networks. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2018*, 283–291.

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

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

Ganian, R.; Hamm, T.; and Mescoff, G. 2020. The Complexity Landscape of Resource-Constrained Scheduling. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, 1741–1747.

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

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

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

Hopcroft, J. E.; and Karp, R. M. 1973. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing* 2(4): 225–231.

Johnson, D. S. 1974. Fast Algorithms for Bin Packing. *Journal of Computer and System Sciences* 8(3): 272–314.

Johnson, D. S.; Demers, A. J.; Ullman, J. D.; Garey, M. R.; and Graham, R. L. 1974. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. *SIAM Journal on Computing* 3(4): 299–325.

Kumar, A.; and Kleinberg, J. M. 2006. Fairness Measures for Resource Allocation. *SIAM Journal on Computing* 36(3): 657–680.

Lang, J.; and Rothe, J. 2016. Fair Division of Indivisible Goods. In Rothe, J., ed., *Economics and Computation, An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division*, Springer Texts in Business and Economics, 493–550. Springer.

Lenstra, J.; and Rinnooy Kan, A. 1980. Complexity results for scheduling chains on a single machine. *European Journal of Operational Research* 4(4): 270 – 275. ISSN 0377-2217. Combinational Optimization.

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

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

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

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

Porschen, S.; Schmidt, T.; Speckenmeyer, E.; and Wotzlaw, A. 2014. XSAT and NAE-SAT of linear CNF classes. *Discrete Applied Mathematics* 167: 1–14.

Sturm, L. B. J. M. 1970. A Simple Optimality Proof of Moore's Sequencing Algorithm. *Management Science* 17(1): 116–118.

Walsh, T. 2020. Fair Division: The Computer Scientist's Perspective. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI 2020*, 4966–4972.