Ben-Gurion University
of the Negev

Danny Hermelin

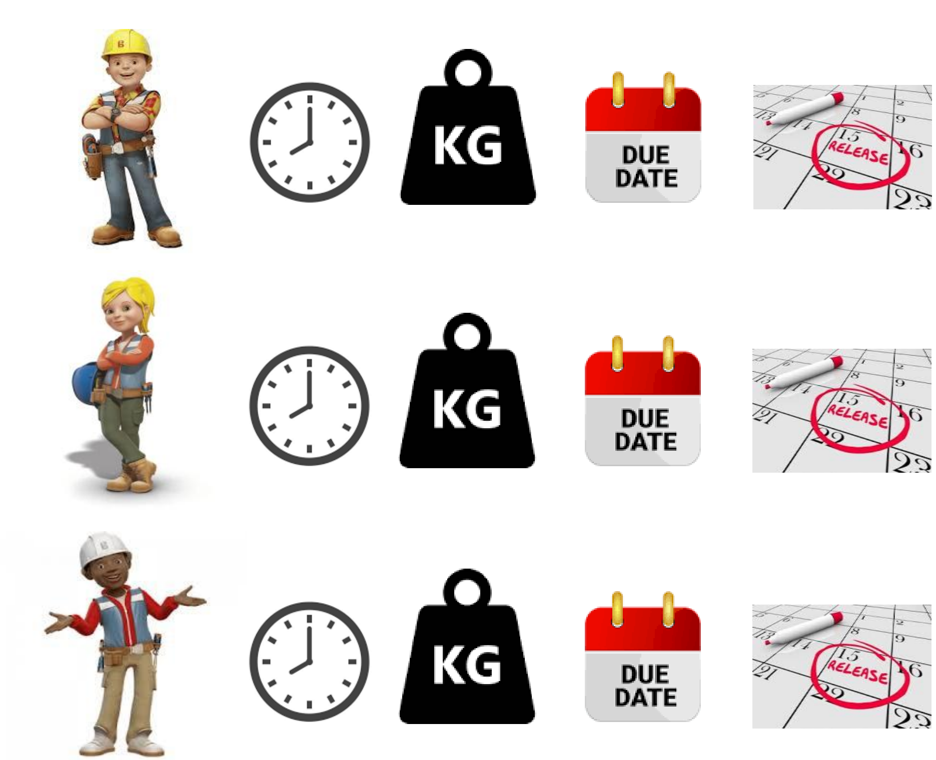
Fairness in Repetitive Scheduling

Joint work with Klaus Heeger, Yuval Itzhaki, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, Michael L. Pinedo, Danny Segev, and Dvir Shabtay

schedulingseminar.com

Classical Scheduling

- Most single machine scheduling problems look like this:
  - There are n jobs (or clients):



- Each job j has:
  - a processing time p_j,
  - a weight w_j (which may equal 1),
  - a due date d_j (sometimes),
  - a release time r_j (sometimes),
  - etc…

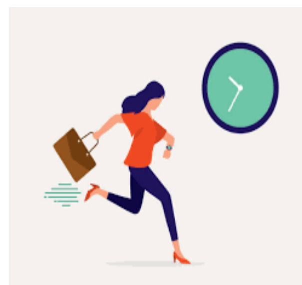
Classical Scheduling

- Most single machine scheduling problems look like this:
  - A schedule (when there are no release times) is simply a permutation of the jobs, specifying the order of processing:

- The completion time of job j is such a schedule is $C_j = \sum_{\substack{i \text{ is not after } j \\ \text{in the schedule}}} p_i$

Classical Scheduling

- The clients may have different objectives:
  - they may want to -



- minimize completion time,

  $$C_j = \sum_{\pi(i) \leq \pi(j)} p_i$$

- minimize lateness,

  $$L_j = C_j - d_j$$

- not be tardy,

  $$U_j = \begin{cases} 1 : & C_j > d_j \\ 0 : & C_j \leq d_j \end{cases}$$

- etc…

Classical Scheduling

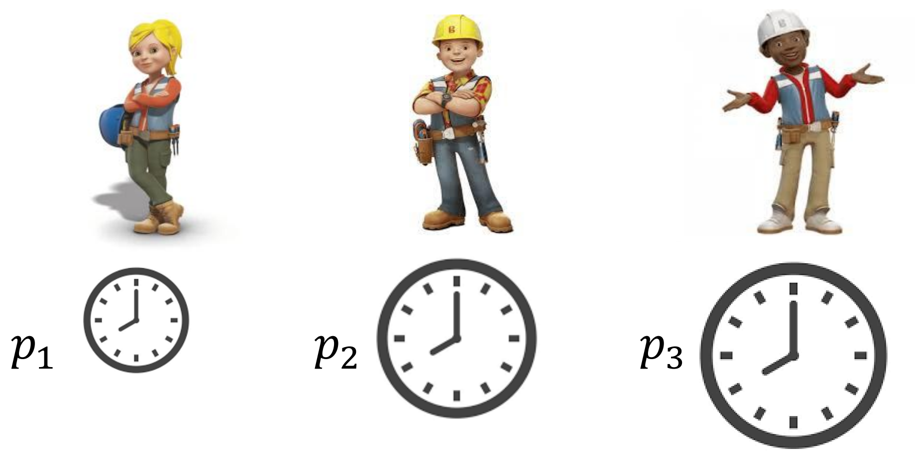
- The scheduler, being the service provider, typically decides the objective of the schedule.

- The scheduler might decide to try to be as fair as possible to all clients.

- One way to do so, is to minimize their total completion time - $\sum_j C_j$
  - which is equivalent to minimizing the average completion time of a client.

Classical Scheduling

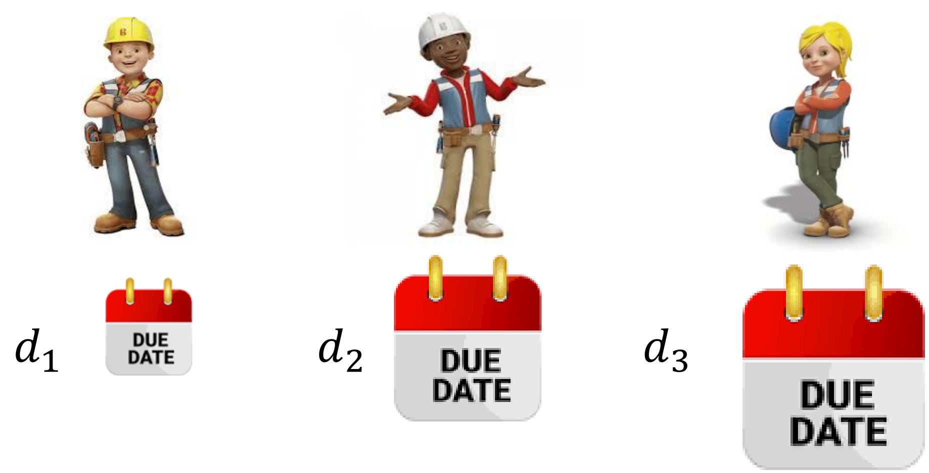
- To minimize ΣC_j, use the Shortest Processing Time first (SPT) rule:



$$\sum_j C_j = 3p_1 + 2p_2 + p_3$$

Classical Scheduling

- Suppose the clients just don’t want to be late.
- To minimize ΣU_j, use the Earliest Deadline First (EDD) rule:

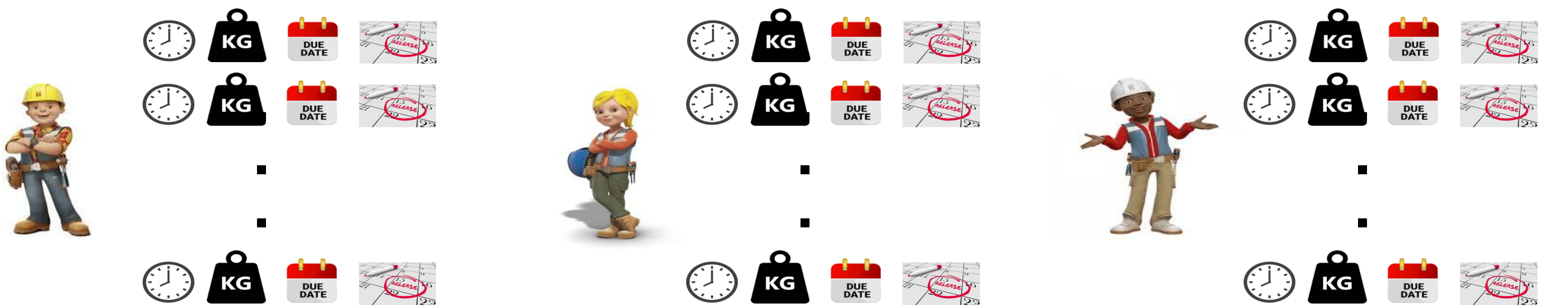


d_1 d_2 d_3

- Scan the jobs in EDD order, and when first encountering a tardy job, remove the largest job already processed (Hodson-Moore Alg.).
- Is this really fair?

Repetitive Scheduling

- But what happens when the clients keep on returning ?



- Now, in the simplest case, each client has m jobs, one per each of the m days of the month.
- Let $p_{i,j}$ denote the processing time of client j's job on day i.
- Define $C_{i,j}$ to be the completion time of client j's job on day.
- Similarly define $w_{i,j}$, $d_{i,j}$, and $r_{i,j}$ when necessary.

Repetitive Scheduling

- If we minimize the total completion time $\Sigma C_{i,j}$, a single client can complete last on every day!

- This is particularly unfair if the job processing time aren't entirely determined by the clients.
- For example:
  - Healthcare.
  - Civic duties (jury duty, military service, ect..).
  - Etc…

Repetitive Scheduling

- If we minimize the total completion time $\Sigma U_{i,j}$, a single client can be tardy every day!

- This means that from his point of view, he never gets any service!

- This is particularly unfair when the job due dates aren't entirely determined by the clients.

Fair Repetitive Scheduling

- Instead, it makes much more sense to minimize

$$\max_{j\in[n]} C_j \quad = \quad \max_{j\in[n]} \sum_{i\in[m]} C_{i,j}$$

in case the clients are interested in minimizing their completion time.

- Or, to minimize

$$\max_{j\in[n]} U_j \quad = \quad \max_{j\in[n]} \sum_{i\in[m]} U_{i,j}$$

in case the clients are interested in not being tardy.

Fair Repetitive Scheduling

- More generally, we may have any (say, minimization) objective function $F_{i,j} = f(C_{i,j},)$ that typically depends on the completion time of the clients on any given day.
- We define the (single machine) Fairness in Repetitive Scheduling problem, $1|rep|max_j\Sigma_i F_{i,j}$, as the problem of minimizing

$$\max_{j\in[n]} F_j \quad = \quad \max_{j\in[n]} \sum_{i\in[m]} F_{i,j}$$

- In the decision version, we are given a fairness threshold K, and the goal is to determine whether there exists a schedule with $\max F_j \leq K$.

Fair Repetitive Scheduling

- Thus, we obtain a performance matrix, which specifies the performance (according to the given objective function) of each job.

$F_{1,1}$	$F_{1,2}$	…	…	$F_{1,n}$
$F_{2,1}$	$F_{2,2}$	…	…	$F_{2,n}$
…	…	…		
…	…	…	…	…
…	…	…	…	…
$F_{m,1}$	$F_{m,2}$	…	…	$F_{m,n}$
F_1	F_2	…	…	F_n

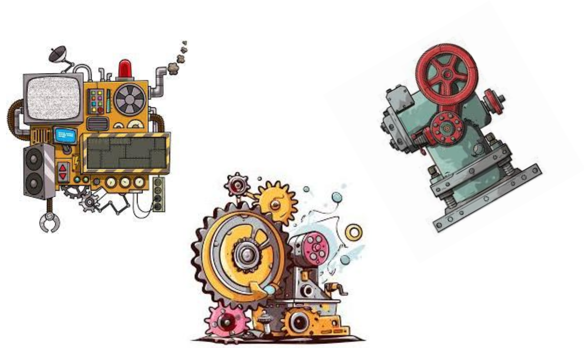
rows correspond to days

column j sums up to F_j

- The goal is to minimize the maximum value in the final row.

Fair Repetitive Scheduling

- Our model is robust, and can handle several tweaks and changes, such as

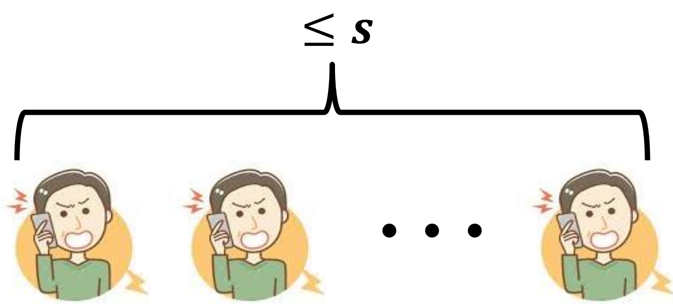


- More complex machine models.
  - E.g., the $Pq|rep|max_j\Sigma_i C_{i,j}$ problem is the problem of minimizing the maximum total completion time of every client when there are q parallel machines every day.

- Different fairness thresholds K_j for each client.
  - E.g., in cases where there are premium customers, etc.

Fair Repetitive Scheduling

- Our model allows several interesting variants to study -



- Is there a schedule which is k-fair for all but s clients?
  - A variant of approximate fairness where a solution is fair to almost everyone.

- The price of fairness?
  - The ratio between the global optimum and the worst k-fair solution.

The $1|rep|\max \Sigma C_{i,j}$ Problem

- Consider first parameter n = number of clients.

Theorem: *The* $1|rep|\max \Sigma C_{i,j}$ *problem is (weakly) NP-hard for two or more clients* ($n \geq 2$).

Theorem: *The* $1|rep|\max \Sigma C_{i,j}$ *problem is pseudo polynomial-time solvable for a constant number of clients* ($n = O(1)$).

- For parameter k = fairness threshold, we can show -

Theorem: *The* $1|rep|\max \Sigma C_{i,j}$ *problem is (strongly) NP-hard even for constant fairness thresholds* ($k \geq 37$).

- Note that this shows that the problem is APX-hard, meaning it doesn't admit a PTAS (most likely).

The 1|rep|max $\Sigma C_{i,j}$ Problem

- Next consider parameter m = number of days. We prove

Theorem: *The* 1|rep|max $\Sigma C_{i,j}$ *problem is (weakly) NP-hard for four or more days* (m≥4).

Theorem: *The* 1|rep|max $\Sigma C_{i,j}$ *problem is polynomial-time solvable for two days* (m=2).

- This leads to the first open problem of the talk:

What is the complexity of 1|rep|max $\Sigma C_{i,j}$ for three days??

The Two Days Algorithm

- The algorithm is inspired by Johnson's algorithm for $F2||C_{max}$.
- We begin with a structural lemma:

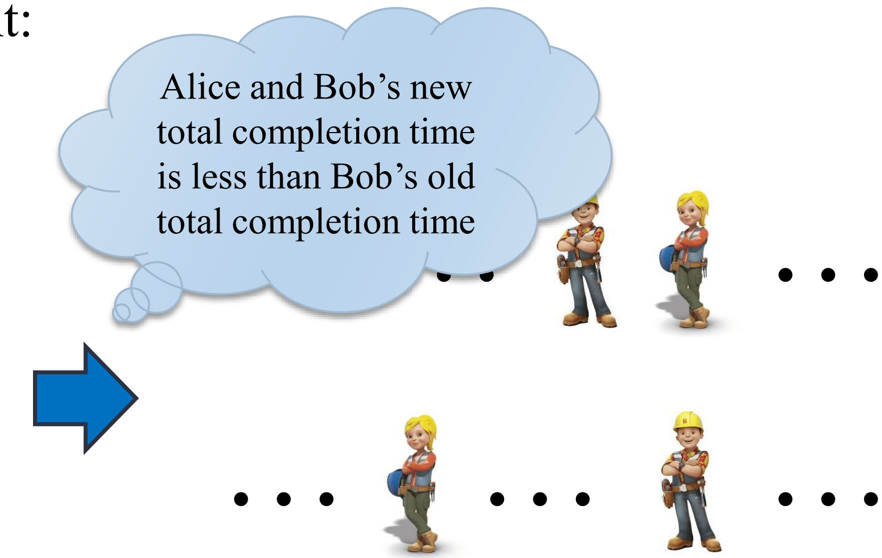
Lemma (Property 1): *There is an optimal schedule for any* $1|rep|\max \Sigma C_{i,j}$ *instance on two days, where the schedule on the second day is in reverse order of the schedule on the first day.*

- Thus, it is enough to only determine the order of processing on the first day.

The Two Days Algorithm

Lemma (Property 1): *There is an optimal schedule for any* $1|rep|\max \Sigma C_{i,j}$ *instance on two days, where the schedule on the second day is in reverse order of the schedule on the first day.*

- The proof is by an exchange argument:



The Two Days Algorithm

- Using the lemma, we construct an optimal schedule as follows:
- Partition the clients into two types:

type 1

smaller processing time on day 1 at least as large as on day 2

$p_{1,j} \leq p_{2,j}$

smaller processing time on day 2

$p_{1,j} > p_{2,j}$

type 2

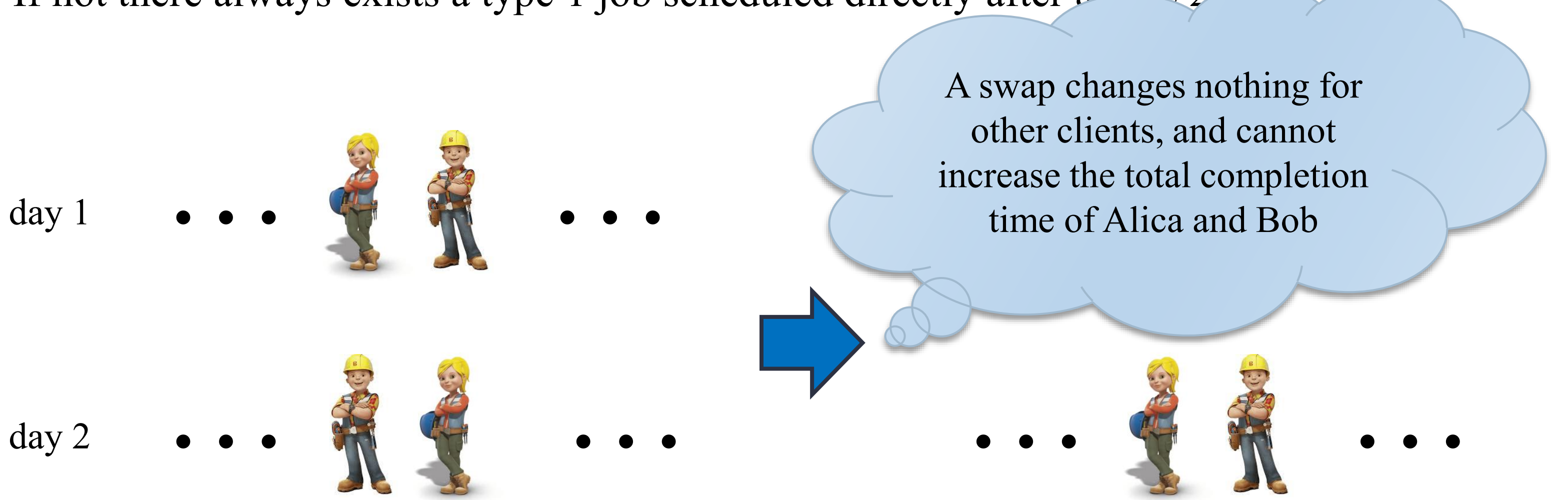
- On day 1: Schedule first the jobs of type 1 clients in SPT order, and then the jobs of type 2 clients in LPT order.
- On day 2 do the reverse of day 1.

The Two Days Algorithm

Lemma: *The schedule constructed is optimal.*

- **<u>Proof:</u>** Consider only optimal solutions where property 1 holds.
  - Step 1: Show that this set includes solutions where type 1 clients are scheduled before type 2 clients on day 1 (Property 2).

If not there always exists a type 1 job scheduled directly after a type 2 j



The Two Days Algorithm

Lemma: *The schedule constructed is optimal.*

- **<u>Proof:</u>** Consider only optimal solutions where property 1 and 2 hold.
  - Step 2: Show that this set only includes solutions where on day 1, type 1 clients are scheduled in SPT order, and type 2 clients scheduled in LPT order (Property 3).

If not, there always is a consecutive pair of clients which n.

A swap changes nothing for other clients, and the new total completion time of Alice and Charlie is at most Alice's old completion time

day 1 • • •  • • • • • •

day 2 • • •  • • • • • •  • • •

Parameterized Complexity

- We can also show the following -

Theorem: *The* $1|rep|\max \Sigma C_{i,j}$ *problem is strongly* W[1]-*hard when parameterized by the number of days.*

- This implies that the problem is unlike to admit a $f(m)n^{O(1)}$ algorithm, even if the processing times are given in unary.
  - A $n^{f(m)}$ algorithm is still possible (yet unknown) in this case.

- One the other hand, we can show that -

Theorem: *The* $1|rep|\max \Sigma C_{i,j}$ *problem can be solved in* $f(K+m)n^{O(1)}$ *time.*

- The proof uses n-fold ILPs.

Approximation Algorithms

- We say that a solution is α-approximate if it has a fairness threshold of k such that $k \leq \alpha \cdot OPT$, where OPT is the optimal fairness threshold.
- We show that -

Theorem: *The* $1|rep|\max \Sigma C_{i,j}$ *problem admits a 2-approximation algorithm that runs in polynomial-time.*

- While the algorithm runs in polynomial-time, it relies on several applications of an LP solver.

Is there a more efficient (combinatorial) constant approximation algorithm??

2-Approximation Algorithm

- The algorithm is inspired by 3-approximation algorithm for $1|r_j|\Sigma w_j C_j$ of Hall, Schulz, D. B. Shmoys, and Wein [MathofOR'97].
- Consider the following LP:

$$
\begin{aligned}
\min \quad & K \\
\text{s.t.} \quad & \sum_{i \in [m]} x_{i,j} \leq K && \forall\, j \in [n] \\
& \sum_{j \in S} p_{i,j} x_{i,j} \geq \frac{1}{2} \cdot P_i^2(S) && \forall\, i \in [m],\ S \subseteq [n]
\end{aligned}
$$

- The variables are:
  - $x_{i,j}$ = completion time of client j's job on day i.
  - K = the fairness threshold.

2-Approximation Algorithm

- The first set of constraints ensures that no client has total completion time which exceeds the fairness threshold k:

$$\sum_{i \in [m]} x_{i,j} \leq K \qquad \forall\, j \in [n]$$

- The second set is less clear -

$$\sum_{j \in S} p_{i,j} x_{i,j} \geq \frac{1}{2} \cdot P_i^2(S) \qquad \forall\, i \in [m],\ S \subseteq [n]$$

  - Here $P_i^2(S)$ denotes the total processing time of the jobs of S on day i, squared.

2-Approximation Algorithm

- The constraint is clearly satisfied for every singleton S={j} since

$$p_{i,j} \cdot C_{i,j} \geq p_{i,j} \cdot p_{i,j} > \frac{1}{2} \cdot p_{i,j}^2 = \frac{1}{2} \cdot P_i^2(S).$$

- And for |S|>1 we have

$$\begin{aligned} \sum_{j \in S} p_{i,j} C_{i,j} \quad &\geq \quad \sum_{j,k \in S, j \leq k} p_{i,j} p_{i,k} \quad = \quad \frac{1}{2} \cdot \Big(\sum_{j \in S} p_{i,j} \Big)^2 + \frac{1}{2} \sum_{j \in S} p_{i,j}^2 \\ &> \quad \frac{1}{2} \cdot \Big(\sum_{j \in S} p_{i,j} \Big)^2 \quad = \quad \frac{1}{2} \cdot P_i^2(S). \end{aligned}$$

- As either the job of client j is scheduled before the job of client k, or vise-versa.
- Thus, every schedule satisfies the second set of constraints.

2-Approximation Algorithm

$$
\begin{aligned}
\min \quad & K \\
\text{s.t.} \quad & \sum_{i \in [m]} x_{i,j} \le K && \forall\, j \in [n] \\
& \sum_{j \in S} p_{i,j} x_{i,j} \ge \frac{1}{2} \cdot P_i^2(S) && \forall\, i \in [m],\ S \subseteq [n]
\end{aligned}
$$

- Note that our LP has an exponential number of constraints, one for each subset of clients.
- Lucky, the LP has a separation oracle (due to Queyranne [MathProg'93]) -
  - A separation oracle is a polynomial-time algorithm that receives a solution x to the LP and determines whether x is feasible or not. If x is not feasible, the oracle returns a constraint which is violated by x.
- One can use the Primal-Dual method, in conjunction with the oracle, to obtain an optimal solution for the LP.

2-Approximation Algorithm

- Let $x^*_{1,1}, \ldots, x^*_{n,m}$ denote an optimal solution for the LP.

- Using this solution, we construct a schedule for each day i by processing the jobs in non-decreasing values of $x^*_{i,j}$.

schedule π_i
for day i

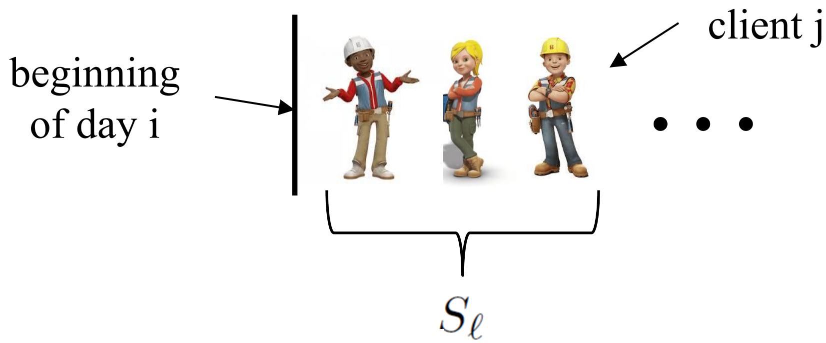
$x^*_{i,j}$ $x^*_{i,j}$ $x^*_{i,j}$

- The completion time of client j's job on day i is $C_{i,j} = \sum_{\pi_i(k) \le \pi_i(j)} p_{i,k}$.

2-Approximation Algorithm

Lemma: *The schedule constructed is 2-approximate.*

- **Proof:** Fix $i \in [m]$ and $j \in [n]$, and consider the job of client j on day i.
- Let S_ℓ denote the set of clients whose jobs were not scheduled after the job of client j (i.e., j and those before him on day i).



- Then, by our construction, we have $C_{i,j} = \sum_{\pi_i(k) \leq \pi_i(j)} p_{i,k} = P_i(S_\ell)$.

2-Approximation Algorithm

Lemma: *The schedule constructed is 2-approximate.*

- **Proof:** Fix $i \in [m]$ and $j \in [n]$, and consider the job of client j on day i.
- Then, by our construction, we have $C_{i,j} = \sum_{\pi_i(k) \leq \pi_i(j)} p_{i,k} = P_i(S_\ell)$.
- On the other hand, as $x^*_{1,1}, \ldots, x^*_{n,m}$ is feasible, and $x^*_{i,j} \geq x^*_{i,k}$ for all clients $k \in S_\ell$, we have

$$\frac{1}{2} \cdot P_i^2(S_\ell) \leq \sum_{k \in S_\ell} p_{i,k} x^*_{i,k} \leq x^*_{i,j} \cdot \sum_{k \in S_\ell} p_{i,k} = x^*_{i,j} \cdot P_i(S_\ell).$$

- It follows that $2x^*_{i,j} \geq P_i(S_\ell) = C_{i,j}$
- Thus, for any client $j \in [n]$, we get $\sum_{i=1}^{n} C_{i,j} \leq 2 \sum_{i=1}^{n} x^*_{i,j} \leq 2K^*$,

  where K^* is the optimal fairness threshold.

Further Results

- We can also show the following -

Theorem: *The* $1|\text{rep}|\max \Sigma C_{i,j}$ *problem admits a* PTAS *for a constant number of days.*

- The PTAS replies on an involved batching scheme where we batch groups of jobs together.
- We also consider the all days are the same case, and show -

Theorem: *The* $1|\text{rep},p_{i,j}=p_j|\max \Sigma C_{i,j}$ *problem admits a* $\frac{(1+\sqrt{2})}{2}$*-approximation algorithm that runs in near linear-time.*

Theorem: *The* $1|\text{rep},p_{i,j}=p_j|\max \Sigma C_{i,j}$ *problem admits a* QPTAS.

The $1|rep|\max \Sigma U_{i,j}$ Problem

- The problem is already quite hard, even when each client has the same due date on each day:

Theorem: *The* $1|rep,d_{i,j}=d_j|\max \Sigma U_{i,j}$ *problem is NP-hard, even for K=1.*

  - In the all days are the same variant, we can show for the single due date case a 1.5-approximation algorithm

- On the other hand, it is easy for unit processing times:

Theorem: *The* $1|rep,p_{i,j}=1|\max \Sigma U_{i,j}$ *problem is polynomial-time solvable.*

  - This theorem holds even when the jobs have release times, and there are multiple parallel machines for processing.

The 1|rep|max $\Sigma Z_{i,j}$ Problem

- This problem is quite hard as well:

Theorem: *The* 1|rep|max $\Sigma Z_{i,j}$ *problem is polynomial-time solvable for* $k \in \{m-1, m\}$*, and NP-hard otherwise.*

- However, when the number of clients is small -

Theorem: *The* 1|rep|max $\Sigma Z_{i,j}$ *problem can be solved in* $f(n)m^{O(1)}$ *time.*

- And when then the number of days is small -

Theorem: *The* 1|rep,di,j=dj|max $\Sigma Z_{i,j}$ *problem can be solved in* $O(mn)^{(m+1)}$ *time.*

- The all days are the same case translates directly to interval graph coloring.

Biblography

1. *Klaus Heeger,* **D.H.**, *George B. Mertzios, Hendrik Molter, Rolf Niedermeier, Dvir Shabtay: Equitable scheduling on a single machine. J. Sched. 26(2): 209-225 (2023).*
2. **D.H.**, *Hendrik Molter, Rolf Niedermeier, Michael L. Pinedo, Dvir Shabtay: Fairness in repetitive scheduling. Eur. J. Oper. Res. 323(3): 724-738 (2025).*
3. *Klaus Heeger,* **D.H.**, *Yuval Itzhaki, Hendrik Molter, Dvir Shabtay: Fair repetitive interval scheduling. Algorithmica 87(9): 1340-1368 (2025).*
4. **D.H.**, *Danny Segev, Dvir Shabtay: Approximate fair repetitive scheduling. Work in progress.*

THANK YOU FOR YOUR ATTENTION