A PTAS for Minimizing Weighted Flow Time on a Single Machine

Alexander Armbruster  
TU of Munich  
Germany  
alexander.armbruster@tum.de

Lars Rohwedder*  
Maastricht University  
Netherlands  
l.rohwedder@maastrichtuniversity.nl

Andreas Wiese  
TU of Munich  
Germany  
andreas.wiese@tum.de

1 INTRODUCTION

A natural and classical scheduling problem is to minimize the sum of weighted flow times of the given jobs (or equivalently, the average weighted flow time). In this paper, we study the setting of a single machine in which we allow the jobs to be preempted (and resumed later). Formally, the input consists of a set of jobs \( J \), where each job \( j \in J \) is characterized by a processing time \( p_j \in \mathbb{N} \), a release time \( r_j \in \mathbb{N} \), and a weight \( w_j \in \mathbb{N} \). We seek to compute a (possibly) preemptive schedule for the jobs in \( J \) on a single machine, which respects the release times, i.e., each job \( j \in J \) is not processed before its release time \( r_j \). For each job \( j \in J \) we denote by \( C_j \) the completion time of \( j \) in the computed schedule, and by \( F_j = C_j - r_j \) its flow time. Our objective is to minimize \( \sum_{j \in J} w_j F_j \).

It was a longstanding open problem to find a polynomial time \( (1+\varepsilon) \)-approximation algorithm for this problem. In a breakthrough result, Batra, Garg, and Kumar found a \( (1+\varepsilon) \)-approximation algorithm in pseudo-polynomial time \([7]\). Their approach is based on a reduction of the problem to the Demand Multicut Problem on Trees in which we are given a tree whose edges have capacities and costs, and paths with demands, and we seek to satisfy the demand of each path by selecting edges from the tree in the cheapest possible way. The reduction to this problem loses a factor of 32, and the resulting instances are solved approximately with a dynamic program and LP rounding. The final approximation ratio was not explicitly stated in \([7]\), but it is at least 10,000 \([13]\). This was then improved to a \( (1+\varepsilon) \)-approximation algorithm in polynomial time by Feige, Kulkarni, and Li \([12]\), who gave a black box reduction to instances with polynomially bounded values for the job’s processing times and weights.

Subsequently, Rohwedder and Wiese improved the approximation ratio to \( 2+\varepsilon \) \([15]\). They reduced the given instances of weighted flow time to a geometric covering problem in which instead of edges of the tree we select axis-parallel rectangles in the plane (which are arranged in a tree-like structure), and instead of the paths in the tree we have vertical downward rays whose demands we need to satisfy (see Figure 1). This reduction loses only a factor of \( 1+\varepsilon \) and they solve the resulting instances up to a factor of \( 2+\varepsilon \) with a dynamic program (DP).

An open question is whether weighted flow time on a single machine admits a PTAS, which is particularly intriguing since a QPTAS has been known for more than 20 years \([10, 12]\). In this paper we answer this question affirmatively.

1.1 Our Contribution

We present a polynomial time \( (1+\varepsilon) \)-approximation algorithm for the (preemptive) weighted flow time problem on a single machine. Our approximation ratio is best possible, unless \( P = NP \), since the problem is strongly NP-hard \([9]\). We use the same reduction as...
Figure 1: Example of the rectangles and rays in the instances of the geometric covering problem to which we reduce weighted flow time.

in [13] to a geometric covering problem. However, we solve the resulting instance of the latter problem exactly in pseudo-polynomial time and hence we obtain an approximation ratio of $1 + \epsilon$ overall. Via the reduction in [12] we finally obtain an $(1 + \epsilon)$-approximation in polynomial time.

The algorithm in [13] uses a DP in order to compute a $(2 + \epsilon)$-approximation for a given instance of the mentioned geometric covering problem, see Figure 1. ThisDP heavily uses the mentioned tree-structure of the rectangles, but it does not use any special properties of the rays. However, these rays and their demands are highly structured. For each interval $[s, t]$ with $s, t \in \mathbb{N}$ there is a ray $L[s, t]$ in the reduced instance. This ray models that some jobs released during $[s, t]$ need to finish after $t$, since our machine can process at most $t - s$ units of work during $[s, t]$. In fact, the demand of $L[s, t]$ is exactly the total processing time of such jobs minus $t - s$. This induces some structural properties. For example, for two rays $L[s, t], L[s', t]$ with $s' < s$ their demands are related: they differ by the total processing time of jobs released during $[s, s')$ minus the term $s' - s$ (and in particular this difference is independent of $t$ and the jobs released during $[s, t]$).

The DP in [13] can be thought of as a recursive algorithm that traverses the tree structure of the rectangles from the root to the leaves. Intuitively, each recursive call corresponds to a vertex of the tree and it decides for some of the rectangles whether they are selected or not. In order to ensure that the demand of each ray is satisfied, we would like to “remember” the previously selected rectangles on the path from the root to the current vertex. This can be done via including this information in the parameters that describe the DP-cell corresponding to the recursive call. However, we cannot afford to remember this information exactly, since this would yield too many DP-cells. The DP in [13] uses complicated techniques such as smoothing and “forgetting” some rectangles (similar to [7]) in order to bound the number of DP-cells by a polynomial. However, it does not remember enough information to compute a $(1 + \epsilon)$-approximation (or even an exact solution), but it loses a factor of $2 + \epsilon$ in the approximation ratio.

Instead, we take advantage of the structure of the rays’ demands and present a different DP that solves our instances of the geometric covering problem exactly in pseudo-polynomial time (so we do not even lose a factor of $1 + \epsilon$). In each DP-cell corresponding to a vertex of the tree, we remember $O(1)$ well-chosen values (the $\epsilon$ stems from the reduction), which intuitively correspond to the maximum deficits up to now from rays that intersect with rectangles corresponding to the current tree vertex but that also intersect with rectangles corresponding to the ancestors of the current vertex. In particular, our algorithm is much more compact and arguably simpler than the DP in [13].

1.2 Other Related Work

Prior to the mentioned first constant factor-approximation algorithm in pseudo-polynomial time [7], Bansal and Pruhs [6] found a polynomial time $O(\log^2 P)$-approximation algorithm for the general scheduling (GSP) problem where $P$ denotes the ratio between the largest and the smallest processing time of jobs in a given instance. GSP subsumes weighted flow time and other scheduling objectives on a single machine. Also, for the special case of weighted flow time where $w_j = 1/p_j$ for each job $j \in J$, i.e., we minimize the average stretch of the jobs, there is a PTAS [8, 10]. Also, there is a PTAS for weighted flow time if $P = O(1)$ [10].

Weighted flow time is also well studied in the online setting in which jobs become known only at their respective release times. For this case, there is a $O(\min(\log W, \log P, \log D))$-competitive algorithm by Bansal and Dhamdhere [3] and a $O(\log^2 P)$-competitive algorithm by Chekuri, Khanna, and Zhu [11]. However, there can be no online $O(1)$-competitive algorithm, as shown by Bansal and Chan [2]. On the other hand, this is possible if the online algorithm can process jobs at a slightly faster rate (by a factor $1 + \epsilon$) than the optimal solution, as Bansal and Pruhs show [4, 5].

There are also FPT-$(1 + \epsilon)$-approximation algorithms known for weighted flow time on one or several machines, where the parameters are the number of machines, an upper bound on the (integral) jobs’ processing times and weights, and $\epsilon$ [14].

2 GEOMETRIC PROBLEM

Rohwedder and Wiese [13] reduce the weighted flow time problem to a two-dimensional geometric covering problem. This reduction
is similar to a reduction by Batra, Garg, and Kumar [7] who reduce weighted flow time to the demand multicut problem on trees. The main difference is that the former reduction is technically more involved, but loses only a factor of $(1+\varepsilon)$ in the approximation ratio. In this section, we summarize the key properties of the reduction in [13].

Let $\varepsilon > 0$ and assume w.l.o.g. that $1/\varepsilon \in \mathbb{N}$. In order to avoid technical difficulties later, before we describe the problem arising from the reduction, we show that we can assume that the release times of the jobs are pairwise distinct, losing only a factor of $1 + \varepsilon$. In the following we write $P := \max_{j \in J} p_j$. We note that in some of the related results mentioned above $P$ is defined as the ratio $\max_{j \in J} p_j / \min_{j \in J} p_j$. However, one can transform any given instance to an instance with new processing times $p'_j$ in which $p'_j \in \{1, 2, \ldots, \lfloor \max_{j \in J} p_j / \min_{j \in J} p_j \rfloor \}$ for each job $j \in J$, while losing only a factor of $1 + \varepsilon$ (see e.g. [15]). Then, the two definitions are essentially the same.

**Lemma 1.** By losing a factor of $1 + \varepsilon$ in the approximation ratio and increasing $P$ by a factor of $n/\varepsilon$, we can assume that $r_j \neq r_j'$ for any two distinct jobs $j, j' \in J$.

**Proof.** Assume that $J = \{1, \ldots, n\}$. For each job $j \in J$ we define a new release time $r'_j := r_j + \varepsilon n$. For these changed release times there is a schedule of cost at most $\text{OPT} + \sum_{j \in J} r w_j \leq \text{OPT} + \sum_{j \in J} r w_j p_j \leq (1 + \varepsilon)\text{OPT}$ which we can obtain by taking OPT and shifting the whole schedule by $\varepsilon n$ units to the right, i.e., delaying every (partial) execution of a job by $\varepsilon n$ units. On the other hand, any schedule $S$ for the new release times $\{r'_j\}_{j \in J}$ yields a schedule for the original release dates $\{r_j\}_{j \in J}$ of cost at most $S + \sum_{j \in J} r w_j \leq S + \varepsilon n S \leq (1 + \varepsilon)S$ (where we write $S$ for the cost of $S$). Finally, we scale up all processing times $\{p_j\}_{j \in J}$ and release times $\{r'_j\}_{j \in J}$ by a factor $n/\varepsilon$ in order to ensure that they are integral again. □

We define $T := \max_j r_j + \sum_j p_j$ and observe that in the optimal solution clearly each job finishes by time $T$ or earlier. Also, we can assume w.l.o.g. that $T \leq n P$ by splitting the given instance if necessary.

In the reduction to our geometric covering problem, we introduce for each job $j$ a set of unit height rectangles, and each rectangle $R$ corresponding to $j$ has a cost $c(R)$ and a capacity $p(R)$. All introduced rectangles are pairwise non-overlapping and all of their vertices have integral coordinates, see Figure 1. Also, they form a tree-like structure, which we will make precise below. For each interval of time $I = [s, t]$ we introduce a vertical (downward-pointing) ray $L(I)$ with a demand $d(I)$. The goal of the geometric covering problem is to select a subset of the rectangles of minimum total cost such that the demands of all rays are covered. The demand of a ray is covered if the total capacity of the selected rectangles intersecting with the ray is at least the demand of the ray. There are additional technical local constraints that restrict what combinations of rectangles can be selected, which we will make precise below.

The rectangles and rays created by the reduction have special structural properties that will later allow us to solve the resulting instances optimally in pseudo-polynomial time. We will describe now all their relevant properties. For a complete description of the reduction we refer to [13].

**Hierarchical Grid.** We subdivide the $x$-axis into hierarchical levels of grid cells. This grid is parameterized by a constant $K = (2/\varepsilon)^{1/\varepsilon}$ (assuming w.l.o.g. $1/\varepsilon \in \mathbb{N}$). Each cell $C$ correspond to an interval $[t_1, t_2]$ with $t_1, t_2 \in \mathbb{N}$ and we define $\text{beg}(C) = t_1$, $\text{end}(C) = t_2$ and $\text{len}(C) = t_2 - t_1$. At level $0$ there is only one cell $C_0$, which contains $[0, T]$. The precise coordinates of the interval end-points are subject to random shifts and we do not define them formally here. The construction ensures that $\text{len}(C_0) \leq K^2 n P = O_p(n P)$.

The cells are organized in a tree such that each cell $C$ of some level $t$ with $\text{len}(C) > K$ has exactly $K$ children of level $t + 1$, and these children form a subdivision of $[\text{beg}(C), \text{end}(C)]$ into $K$ subintervals of length $\text{len}(C)/K$ each. Each cell $C$ with $\text{len}(C) \in \{1, 2, \ldots, K\}$ does not have any children. For a cell $C$ we denote its level by $t(C)$. Let $t_{\max}$ denote the maximum level of the cells and let $C$ denote the set of all cells.

**Segments.** Each job $j$ is associated with segments $\text{Seg}(j)$ that partition the interval $[r_j, \text{end}(C_0)]$ (recall that $C_0$ is the unique cell at level $0$). More precisely, there is a set of cells that cover $[r_j, \text{end}(C_0)]$, one cell from each level $t \in \{0, \ldots, t_{\max}\}$. Then for each of these cells $C$ we define a set of segments $\text{Seg}(C, j)$. Each segment $S \in \text{Seg}(j, C)$ is contained in $C$. Also, if $t(C) \leq t_{\max} - 2$ then $S$ is a cell of level $t(C) + 2$, and if $t(C) \in \{t_{\max} - 1, t_{\max}\}$ then $S$ is an interval $[t, t + 1]$ for some $t \in \mathbb{N}$. In particular, the segments in $\text{Seg}(j, C)$ all have the same length. If $t(C) \leq t_{\max} - 1$ then the the number of segments in $\text{Seg}(j, C)$ is an integral multiple of $K$ in $\{K, 2K, \ldots, K^2\}$ and if $t(C) = t_{\max}$ there are at most $K$ segments in $\text{Seg}(j, C)$. Furthermore, their union forms an interval ending in $\text{end}(C)$. If $t(C) \leq t_{\max} - 1$ this interval starts at $\text{beg}(C')$ for a child $C'$ of $C$. Otherwise, $t(C) = t_{\max}$ and then this interval starts at some integral value in $C$. Please check this, I added this. The set $\text{Seg}(j)$ is the union of all segments in the sets $\text{Seg}(C, j)$ for all cells $C \in C$ and $\text{Seg}(j)$ has the property that the segment lengths are non-decreasing from left to right.

The construction of the segments guarantees the following relationship between segments of different jobs: Let $j, j' \neq \text{jobs}$ be jobs with $r_j \leq r_{j'}$ and consider a group of segments $\text{Seg}(j', C')$. Then there exists a cell $C$ such that the interval spanned by $\text{Seg}(j', C')$ is contained in the interval spanned by $\text{Seg}(j, C)$, i.e., $\bigcup_{S' \in \text{Seg}(j', C')} S' \subseteq \bigcup_{S \in \text{Seg}(j, C)} S$, and in addition $C' = C$ or $C'$ is a descendant of $C$; see also Figure 2.

**Rectangles.** We are now ready to describe the rectangles. First, assume that the jobs correspond to integers $1, 2, \ldots, n$ and they are sorted increasingly by their release times. Consider a job $j$. For each segment $S \in \text{Seg}(j)$ there is a rectangle $[\text{beg}(S), \text{end}(S)] \times [j, j + 1]$, see Figure 3 (in our figures we assume that the origin is in the top left corner and that the $x$- and $y$-coordinates increase when going to the right and down, respectively). For each cell $C \in C$ we define $\mathcal{R}(j, C)$ as the set of all rectangles derived from segments in $\text{Seg}(j, C)$. Also, we define $\mathcal{R}(j) := \bigcup_{C \in C} \mathcal{R}(j, C)$ and $\mathcal{R} := \bigcup_{j \in J} \mathcal{R}(j)$. Each rectangle $R \in \mathcal{R}(j)$ has a cost $c(R) \in \mathbb{N}$ and a capacity $p(R)$ with $p(R) = p_j$. For any set of rectangles $\mathcal{R}' \subseteq \mathcal{R}$ we define $\bar{p}(\mathcal{R}') := \sum_{R \in \mathcal{R}'} p(R)$. 

1337
3 DYNAMIC PROGRAM

The slightly over-simplified idea of our dynamic program is the following. We have an entry in our DP-table (i.e., a DP-cell) for each combination of a job $j$ and a cell $c$ such that $R(j, c) \neq \emptyset$. Let $S_L$ and $S_R$ denote the leftmost and rightmost segments in $Seg(j, c)$ respectively. Our goal is to select rectangles contained in the region $A = \{beg(S_L), end(S_R)\} \times [j, \infty)$ (see Figure 4) such that we satisfy the demand of all rays that are contained in $A$. In this DP-cell, we want to store the solution of minimum cost with this property. In order to compute the solution for this DP-cell, we try out all possibilities for selecting rectangles in $R(j, c)$, obeying the rule that we need to select a prefix of them. Then we combine this choice with the solutions to one or more DP-cells for the next job $j + 1$ (where the precise definition of these DP-cells is induced by the tree structure in which the rectangles are organized).

If we tried to use this approach by itself there would be a fundamental flaw: suppose for simplicity that the problem is reduced to exactly one DP-cell for $(j + 1, c)$ which corresponds to the region $A' = \{beg(S_L), end(S_R)\} \times [j + 1, \infty)$; then the rectangles in $R(j + 1, c)$ are directly underneath the rectangles in $R(j, c)$, like in Figure 4. There may be rays that are contained in $A$, but not in $A'$, see the red rays in Figure 4. In order to cover their demands, it is relevant which rectangles in $A \setminus A'$ we have chosen. However, so far the idea above does not take these rays into account when choosing the solution inside $A'$.

Intuitively, one approach to fix this problem is to “remember” the remaining demand for each ray that intersects $A'$ but which is not contained in $A'$. By “remember” we mean that we include this information in the parameters of the DP-cell corresponding to $A'$. However, there is a polynomial number of rays in total and a pseudo-polynomial number of possibilities for the remaining demand of each ray. Thus, there is an exponential number of combinations for these remaining demands and we cannot afford to have a DP-cell for each of these combinations.

Instead, we show that we can artificially increase the demands of certain rays contained in $A'$ by at most $K^2$ different amounts, such that any solution satisfies these increased demands if and only if it satisfies also the remaining demands of the mentioned rays that are not contained in $A'$. It will suffice to remember only $K^2 = O(1)$ values to describe these increased demands which yields only a pseudo-polynomial number of possibilities. Therefore, the total number of needed DP-cells is only pseudo-polynomial.

3.1 Definition of the DP-Table

Each entry in our DP-table corresponds to a specific subproblem. To avoid ambiguities, we will always use the term “DP-cell” for the entries of our DP-table and we will use “cell” only for the elements of $C$. Intuitively, each DP-cells corresponds to an infinitely large area $A$ with axis-parallel boundary edges (see Figure 4) described by

- a top edge whose $y$-coordinate corresponds to a job $j \in J$,
- a right edge whose $x$-coordinate is the end-point $end(C)$ of some cell $C$,
- a left edge whose $x$-coordinate is the start-endpoint $beg(C')$ of some child $C'$ of $C$.

Figure 2: Alignment of segments $Seg(j)$ and $Seg(j')$ corresponding to jobs $j$ and $j'$. Small separators indicate end of a segment; big separators indicate end of a group of segments, e.g., $Seg(j, C)$.

Figure 3: Alignment of rectangles corresponding to the segments in Figure 2 for the two jobs $j$, $j'$. The groups, e.g., $R(j, C)$, are shaded alternatingly.

Local Selection Constraint. As mentioned above, our goal is to select a subset of the rectangles in $R$. For this, we require a local selection constraint for the set $R(j, c)$ for each job $j \in J$ and each cell $c \in C$ with $R(j, c) \neq \emptyset$. We require that the solution selects a prefix of the rectangles $R(j, c)$, i.e., that we require the union of the selected rectangles in $R(j, c)$ forms a rectangle that includes the leftmost rectangle in $R(j, c)$. Each such constraint is local in the sense that it affects only one set $R(j, c)$.

Rays. For each interval $I = [s, t]$ with $s, t \in \mathbb{N}$ and $0 \leq s \leq t \leq T$ we introduce a ray $L(I)$. Let $j(I)$ be the job with minimum $r_j$ such that $s \leq r_j$. We define $L(I) = \{t + \frac{1}{2} \times j(I) + \frac{1}{2}, \infty\}$, (see Figure 1) and we define the demand of $L(I)$ as $d(I) = \sum_{j:s\leq r_j,s.t} p_j - (t - s)$. Also, denote by $R(I)$ the rectangles that are intersected by $L(I)$. Let $L$ denote the set of all introduced rays.

The goal of our geometric covering problem is to select a subset $R' \subseteq R$ of the rectangles in $R$ such that for each ray $L(I) \in L$ its demand is satisfied by $R'$, i.e., $p(R' \cap R(I)) \geq d(I)$. In [13] it was shown that any pseudo-polynomial time (approximation) algorithm for this problem yields a pseudo-polynomial time approximation algorithm for weighted flow time, losing only a factor of $1 + \epsilon$ in the approximation ratio. Moreover, Feige, Kulkarni, and Li [12] showed that by losing only a factor of $1 + \epsilon$ in the approximation guarantee, we can assume that the jobs’ processing times and weights of a given instance of weighted flow time are polynomially bounded integers. These reductions yield the following lemma.

Lemma 2 ([12], [13]). Given a pseudo-polynomial time $c$-approximation algorithm for the geometric covering problem defined above (as a black-box), we can construct a polynomial time $(1 + \epsilon)$-approximation algorithm for preemptive weighted flow time on a single machine for any constant $\epsilon > 0$. 

1338
The goal of this subproblem is to select some of the rectangles contained in $A$ in order to satisfy the demand of all rays contained in $A$. For some of these rays the demand is increased according to additional parameters of the DP-cells. For each grandchild $C'$ of $C$, there is one such parameter $f_{C'}$. We increase the demand of a ray $L([s,t])$ by $f_{C'}$ if and only if (i) $s = r_j$, i.e., the $y$-coordinate of the start-point of $L([s,t])$ corresponds to $j$, and (ii) the vertical coordinate of the start-point of $L([s,t])$ is contained in $C'$. Formally, we have a DP-cell for each combination of

- a job $j \in J \cup \{n+1\}$,
- a cell $C' \in C$,
- a value $k \in \{1,\ldots,K\}$ which intuitively indicates that we are only interested in rectangles whose corresponding segments are contained in the $k$-th or $K$-th children of $C$,
- $j$, $C$, and $k$ induce an area $A(j, C, k)$ defined as follows
  - if $l(C) < \tau_{max}$ then $A(j, C, k) := \{\text{beg}(C'), \text{end}(C) \times [j, \infty) \}$ where $C'$ is the $k$-th child of $C$.
  - if $l(C) = \tau_{max}$ then $A(j, C, k) := \{\text{beg}(C) + k - 1, \text{end}(C) \times [j, \infty) \}$

We require that the set $A(j, C, k)$ is nonempty, otherwise the DP-cell is not defined. Note that this happens if and only if $l(C) = \tau_{max} + 1$ and $k > \text{end}(C) - \text{beg}(C)$.

A value $f_{C'} \in \{0, \ldots, \sum_{j' \in j} p_{r_j}\}$ for each cell $C' \in C(C, k)$ where $C(C, k)$ is defined as follows:

- if $l(C) \leq \tau_{max} - 2$ then $C(C, k)$ contains all grandchildren of $C$ that are children of the $k$-th to $K$-th children of $C$.
- if $l(C) = \tau_{max} - 1$ then $C(C, k)$ contains each interval $[t, t+1]$ with $t \in \mathbb{N}$ such that $[t, t+1]$ is contained in the $k$-th child of $C$ for some $k' \in \{k, \ldots, K\}$ (one may think of $[t, t+1]$ being a cell of some dummy level $\tau_{max} + 1$).
- if $l(C) = \tau_{max}$ then $C(C, k)$ contains each interval $[t, t+1]$ with $t \in \mathbb{N}$ and $[t, t+1] \subseteq \{\text{beg}(C) + k - 1, \text{end}(C)\}$.

We require for each job $j'$ and each cell $C'$ that either all rectangles in $R(j', C')$ are contained in $A(j, C, k)$ and that in this case $C'$ is a descendant of $C$ or $C = C'$, or that none of the rectangles in $R(j', C')$ intersect with $A(j, C, k)$ (otherwise the DP-cell $(j, C, k, \{f_{C'}\}_{C'})$ is not defined).

Fix such a DP-cell $(j, C, k, \{f_{C'}\}_{C'})$. We define $\mathcal{R}(\{(j, C, k) \in \mathcal{R} \mid \mathcal{R} \subseteq A(j, C, k)\})$, see Figure 4. The subproblem corresponding to $(j, C, k, \{f_{C'}\}_{C'})$ is to select a set of rectangles $\mathcal{R}' \subseteq \mathcal{R}$ such that

- $\mathcal{R}' \subseteq \mathcal{R}(j, C, k)$,
- for each job $j' \in J$ and each cell $C' \in C$ the set $\mathcal{R}' \cap \mathcal{R}(j', C')$ forms a prefix of $\mathcal{R}(j', C')$,
- for each ray $L([s,t]) \in L$ for some interval $I$ with $L([s,t]) \subseteq A(j, C, k)$ its demand $d(I)$ is covered by $\mathcal{R}'$, i.e., $p(\mathcal{R}' \cap \mathcal{R}(I)) \geq d(I)$,
- for each cell $C' \in C(C, k)$ and each ray $L([r_j, t]) \in L$ with $t \in C'$ (and for which hence $L([r_j, t]) \subseteq A(j, C, k)$ holds) we have even that $p(\mathcal{R}' \cap \mathcal{R}([r_j, t])) \geq d([r_j, t]) + f_{C'}$; for the case $j = n+1$ we define $r_{n+1} := t + 1$.

The objective is to minimize $c(\mathcal{R}')$. In the DP-cell $(j, C, k, \{f_{C'}\}_{C'})$ we store the optimal solution to this sub-problem, or the information that there is no feasible solution for it. In the former case, we denote by $\text{OPT}(j, C, k, \{f_{C'}\}_{C'})$ the optimal solution to the DP-cell.

### 3.2 Filling in the DP-Table

We describe now how to fill in the entries of the DP-table. For each rectangle $R \in \mathcal{R}$ we define by $\text{proj}_x(R)$ its projection to the $x$-axis. Consider a DP-cell $(j, C, k, \{f_{C'}\}_{C'})$ and assume that it has a feasible solution. Let $A := A(j, C, k)$. The base case arises when $A$ does not contain any rectangles from $\mathcal{R}$ (this holds for example when $j = n+1$). In this case we store the empty solution in $(j, C, k, \{f_{C'}\}_{C'})$. This is justified due to the following lemma since $A$ can contain only rays $L([s,t])$ with $R([s,t]) = \emptyset$ and the last condition of the definitions of the sub-problem applies to such a ray only if $s = r_j$ for some job $j'$.

**Lemma 3.** Let $L([s, t]) \in L$ be a ray with $R([s, t]) = \emptyset$. Then $d([s, t]) \leq 0$ and $s$ is not the release time of any job.

**Proof.** Recall that the demand satisfies $d([s, t]) = \sum_{i \leq r, i \leq t} p_i$; suppose the contradiction that there exists a job $j$ with $s \leq r_j \leq t$. Because we have a ray for $[s, t]$ it follows that $t \leq \text{end}(C_0)$. The segments corresponding to $j$ partition the interval
To compute an optimal solution for the corresponding (canonical) DP-cell, we guess the solution for the gray rectangles and look up an appropriate solution corresponding to the hatched area without these gray rectangles, i.e., shifting the upper border of $A$ down by one unit.

Suppose for the remainder of this section that we are given a DP-cell $(j, C, k, \{f_{C'}^{*}\}_{C'})$ for which the corresponding area $A$ contains at least one rectangle from $R$. Also, assume that we have already computed the optimal solution for each cell $(j', C', k', \{f_{C'}^{*}\}_{C'})$ with $j' > j$.

In our next case we will handle what we call canonical DP-cells (see Figure 5). We say that $(j, C, k, \{f_{C'}^{*}\}_{C'})$ is a canonical DP-cell if it satisfies the following properties:

- $R(j, C) \neq \emptyset$,
- for each rectangle $R \in R(j, C)$ we have that $R \not\subseteq A$, and
- the leftmost $x$-coordinate of $A$ coincides with the leftmost $x$-coordinate of the leftmost rectangle in $R(j, C)$.

Note that due to this specification also the rightmost $x$-coordinate of $A$ coincides with the rightmost $x$-coordinate of the rightmost rectangle in $R(j, C)$.

Suppose we are given a canonical DP-cell $(j, C, k, \{f_{C'}^{*}\}_{C'})$. Let $R^*$ be the optimal solution to $(j, C, k, \{f_{C'}^{*}\}_{C'})$ and define $R^*(j, C) := R^* \cap R(j, C)$. We guess $R^*(j, C)$ for which there are at most $O(K^2) = O_1(1)$ possibilities. Next, we define values $f_{C'}^{*}$ for each $C' \in C(C, k)$ there is one rectangle $R_{C'} \in R(j, C)$ with $\text{proj}_x(R_{C'}) = C'$. The value $f_{C'}^{*}$ depends on whether $R_{C'} \in R^*(j, C)$ or not. We set

$$f_{C'}^{*} := \max\{0, f_{C'} + p_j - r_{j+1} + r_j - p(R^*(j, C) \cap \{R_{C'}\})\}.$$  

We store in the cell $(j, C, k, \{f_{C'}^{*}\}_{C'})$ the solution $R^*(j, C) \cup \text{OPT}(j+1, C, k, \{f_{C'}^{*}\}_{C'})$. This is justified due to the following lemma.

**Lemma 4.** Suppose that $(j, C, k, \{f_{C'}^{*}\}_{C'})$ is a canonical DP-cell. Let $f_{C'}^{*}$ be defined as above. Then the DP-cell $(j+1, C, k, \{f_{C'}^{*}\}_{C'})$ exists, it has a feasible solution, and $R^*(j, C) \cup \text{OPT}(j+1, C, k, \{f_{C'}^{*}\}_{C'})$ is an optimal solution for the DP-cell $(j, C, k, \{f_{C'}^{*}\}_{C'})$.

We refer to Section 3.3 for a proof of the lemma. Non-canonical DP-cells are easier to handle in the sense that we do not need to guess anything but simply reduce the problem to one or two other DP-cells. Intuitively, either we reduce the problem to one DP-cell corresponding to the same area $A$ as the original DP-cell, or we partition $A$ into two areas $A_1, A_2$ and reduce the problem to two DP-cells that correspond to these areas.

Suppose now that $(j, C, k, \{f_{C'}^{*}\}_{C'})$ is a non-canonical DP-cell (i.e., not a canonical cell) and that $t(C) < t_{\max}$, see Figure 6. Let $C(k')$ be the $k'$-th child of $C$. For each $C'' \in C(C(k'), 1)$ we define a value $g_{C''}^{*}$ as follows: we identify the (unique) cell $C' \in C(C, k)$ with $C'' \subseteq C'$ and we define $g_{C''}^{*} := f_{C'}$. Also, we define $\{f_{C'}^{*}\}_{C'}$ to be the restriction of $\{f_{C'}^{*}\}_{C}$ to the intervals in $C(C(k'), 1)$. If $k' < K$ then we store in $(j, C, k, \{f_{C'}^{*}\}_{C'})$ the solution $\text{OPT}(j, C(k'), 1, \{g_{C''}^{*}\}_{C''}) \cup \text{OPT}(j, C, k, 1, \{f_{C''}^{*}\}_{C''})$. If $k = K$ we simply store the solution $\text{OPT}(j, C(k'), 1, \{g_{C''}^{*}\}_{C''})$. This is justified by the following lemma.

**Lemma 5.** Assume that $(j, C, k, \{f_{C'}^{*}\}_{C'})$ is a non-canonical DP-cell and that $t(C) < t_{\max}$. Then $(j, C(k'), 1, \{g_{C''}^{*}\}_{C''})$ is a DP-cell and if $k < K$ then $(j, C, k, 1, \{f_{C''}^{*}\}_{C''})$ is a DP-cell as well. Also,

- if $k < K$ then $\text{OPT}(j, C(k'), 1, \{g_{C''}^{*}\}_{C''}) \cup \text{OPT}(j, C, k, 1, \{f_{C''}^{*}\}_{C''})$ is an optimal solution for $(j, C, k, \{f_{C'}^{*}\}_{C'})$.
- if $k = K$ then $\text{OPT}(j, C(k'), 1, \{g_{C''}^{*}\}_{C''})$ is an optimal solution for $(j, C, k, \{f_{C'}^{*}\}_{C'})$.

For the proof see Section 3.4. Finally, suppose that $(j, C, k, \{f_{C'}^{*}\}_{C'})$ is a non-canonical DP-cell and that $t(C) = t_{\max}$, see Figure 7. We define $\{f_{C'}^{*}\}_{C'}$ to be the restriction of $\{f_{C'}^{*}\}_{C}$ to the intervals in $C(C, k+1)$. We store in $(j, C, k, \{f_{C'}^{*}\}_{C'})$ the solution $\text{OPT}(j, C, k+1, 1, \{f_{C''}^{*}\}_{C''})$. This is justified by the following lemma.

**Lemma 6.** Assume that $(j, C, k, \{f_{C'}^{*}\}_{C'})$ is a non-canonical DP-cell and that $t(C) = t_{\max}$. Then $k < K$, $(j, C, k, 1, \{f_{C''}^{*}\}_{C''})$ is
a DP-cell and $\text{OPT}(j, C, k + 1, \{f_{C}^{\ast}\})$ is an optimal solution for $(j, C, k, \{f_{C}^{\ast}\})$.

The proof is given in Section 3.5. Finally, we output the solution in the cell $(1, C_0, 1, \{f_{C}^{\ast}\})$ with $f_{C}^{\ast} = 0$ for each $C$. Our dynamic program yields the following lemma.

Lemma 7. We can compute an optimal solution to an instance of our geometric covering problem in time $(nP)^{O(K^2)} \leq (nP)^{O(1)}$.

Proof. The number of cells in the hierarchical decomposition is bounded by $O(K^2nP)$. For each DP-cell, there are at most $(nP)^K$ many values for $\{f_{C}^{\ast}\}$ to be considered. Thus, the number of DP-cells $(j, C, k, \{f_{C}^{\ast}\})$ is at most $O(nK^2nP\cdot (nP)^K) \leq (nP)^{O(K^2)}$.

We fill in the DP-cells starting with those of the base case as defined above and store the empty solution in each of them. Then, we compute the entries of the DP-cells in decreasing lexicographic order of $(j, f(C), k)$ as described above. This is justified by Lemmas 4-6. Note that in some cases, our constructed (candidate) solution is infeasible for the respective given DP-cell. In this case, we store that the given DP-cell does not have a feasible solution. Guessing the necessary values and applying the rules above can be done in a running time which is polynomial in $n$, $P$, and $K$. Thus, our overall running time is $(nP)^{O(K^2)}$.

Together with Lemma 2 this yields our main theorem.

Theorem 8. There is a polynomial time $(1 + \epsilon)$-approximation algorithm for weighted flow time on a single machine when preemptions are allowed.

3.3 Proof of Lemma 4

Assume that we are given a canonical DP-cell $(j, C, k, \{f_{C}^{\ast}\})$. We split up the proof into multiple steps. First note that a DP-cell can only be canonical if $j \leq n$, because $R(j, C) \neq 0$.

Lemma 9. The DP-cell $(j + 1, C, k, \{f_{C}^{\ast}\})$ exists.

Proof. The only nontrivial part about the conditions for existence (see Section 3.1) are the upper bounds on $f_{C}^{\ast}$ and the last property. Let $C' \in \mathcal{C}(C, k)$, $f_{C}^{\ast} \leq f_{C} + p_j$ and $f_{C'} \leq \sum_{j' < j} p_{j'}$, we obtain $f_{C'}^{\ast} \leq \sum_{j' < j} p_{j'} + f_{C}^{\ast}$, so $f_{C'}^{\ast}$ is within the given bounds. As $A(j, C, k) \cap A(j + 1, C, k) \subseteq \mathbb{R} \times [j, j + 1]$, a rectangle $R \in R(j', C')$ for a job $j'$ and a cell $C'$ with $R \subseteq A(j, C, k)$ is contained in $A(j + 1, C, k)$ if and only if $j' \geq j + 1$. This implies that a rectangle $R \in R(j', C')$ is contained in $A(j + 1, C, k)$ if and only if $j' \geq j + 1$ and $R \subseteq A(j, C, k)$. As $(j, C, k, \{f_{C}^{\ast}\})$ is a DP-cell, either all rectangles in $R(j', C')$ are contained in $A(j, C, k)$ or none of them intersect $A(j, C, k)$. This implies that either all rectangles in $R(j', C')$ are contained in $A(j + 1, C, k)$ or none of them intersect $A(j + 1, C, k)$. It follows that the DP-cell $(j + 1, C, k, \{f_{C}^{\ast}\})$ exists.

Lemma 10. The set $\mathcal{R}'' := \mathcal{R}'' \cap \mathcal{R}_1(j + 1, C, k)$ is a feasible solution for $(j + 1, C, k, \{f_{C}^{\ast}\})$.

Proof. The first three properties of $\mathcal{R}''$ being feasible (see Section 3.1) follow directly from the fact that $\mathcal{R}''$ is feasible for $(j, C, k, \{f_{C}^{\ast}\})$. For the fourth property, let $C' \in \mathcal{C}(C, k)$, $t \in C'$ and suppose that there is a ray $L([r_j, t])$ as there is nothing to show if the ray does not exist. Then also the ray $L([r_j, t])$ exists. As $\mathcal{R}''$ is feasible for $(j, C, k, \{f_{C}^{\ast}\})$, we know that the ray $L([r_j, t])$ is covered by $\mathcal{R}''$, i.e. $p(R'' \cap \mathcal{R}([r_j, t])) \geq d([r_j, t]) + f_{C}^{\ast}$. As $\mathcal{R}([r_j + 1, t]) \cup \mathcal{R}_{C'} = \mathcal{R}([r_j, t])$, it follows that

\[
(p(R'' \cap \mathcal{R}([r_j, t])))
\]

As all release times are different, it follows from Lemma 1 that $L([r_j + 1, t]) \subseteq A(j + 1, C, k)$. Hence, the third property yields $p(R'' \cap \mathcal{R}([r_j, t])) \geq d([r_j + 1, t])$, which together with the inequality above implies that

\[
(p(R'' \cap \mathcal{R}([r_j, t])))
\]

Together with Lemma 2 this yields our main theorem.

Theorem 8. There is a polynomial time $(1 + \epsilon)$-approximation algorithm for weighted flow time on a single machine when preemptions are allowed.
So if we cover the demand of \( [r_j, t] \) we also cover the demand of \([s, t]\). For this reason we only show that the demands with \( s = r_j \) are covered.

As \( t \geq r_j + 1 \) there is a ray \( L([r_j + 1, t]) \) and since all release times are different by Lemma 1 it follows \( L([r_j + 1, t]) \subseteq A(j + 1, C, d). \) For this ray it holds that

\[
p(OPT(j + 1, C, k, \{f_{C'}\}) \cap R([r_j + 1, t])) \geq d(R([r_j + 1, t])) + f_{C'}.
\]

The ray \( L([r_j + 1, t]) \) intersects with a rectangle \( R \in R(j, C) \), and only if \( t \in \text{proj}_x(R) \). There is a unique \( C' \in C(C, k) \) with \( t \in C' \) and this \( C' \) is the projection of a rectangle \( R_{C'} \in R(j, C) \). Thus, \( t \in \text{proj}_x(R) \) is equivalent to \( R = R_{C'} \). This implies

\[
p(R' \cap R([r_j, t])) = p(OPT(j + 1, C, d, \{f_{C'}\}) \cap R([r_j, t])) + p(R' \cap R([r_j, t]))
\]

\[
= p(OPT(j + 1, C, d, \{f_{C'}\}) \cap R([r_j, t])) + p(R' \cap \{R_{C'}\})
\]

\[
\geq d([r_j, t]) + f_{C'} + p(R'(j, C) \cap \{R_{C'}\})
\]

\[
= d([r_j, t]) - p_j + r_j + r_j - r + f_{C'}
\]

Since we have that

\[
c(R') = c(R'(j, C)) + c(OPT(j + 1, C, k, \{f_{C'}\}))
\]

\[
\leq c(R'(j, C)) + c(R'^{\prime}) = c(R'^{\prime})
\]

and \( R' \) is optimal, we conclude that \( R' \) is also an optimal solution for the DP-cell \( (j, C, k, \{f_{C'}\}) \). This completes the proof of Lemma 4.

### 3.4 Proof of Lemma 5

We split up the proof into several lemmas. Assume that we are given a non-canonical DP-cell \( (j, C, k, \{f_{C'}\}) \) and that \( t(C) < \ell_{\max} \).

**Lemma 12. The union of the projections of \( R(j, C) \) to the x-axis forms an interval \( I = [\text{end}(C(i)), \text{end}(C))] \), where \( C(i) \) is the \( i \)-th child of \( C \) and \( i \geq k \) for some \( i \).**

**Proof.** The projections of the rectangles \( R(j, C) \) are the segments \( \text{Seg}(j, C) \) and the union of these segments is by definition a possibly empty interval \( I \) ending in \( \text{end}(C) \). This interval starts with the leftmost \( x \)-coordinate of the leftmost rectangle in \( R(j, C) \) if \( R(j, C) \neq \emptyset \) and is empty otherwise. Let \( s \) be the left endpoint of \( I \) if \( R(j, C) \neq \emptyset \) and let \( s = \text{end}(C) \) if \( R(j, C) = \emptyset \). Then \( I = [s, \text{end}(C)] \). We need to show that \( s \) is the endpoint of a child cell of \( C \). If \( R(j, C) = \emptyset \) then as \( \text{end}(C) = \text{end}(C') \) we get \( I = [s, \text{end}(C')] \). So for now consider \( R(j, C) \neq \emptyset \). As \( t(C) < \ell_{\max} - 1 \) the structure of the segments implies that there is a value \( i' \in [K] \) such that \( s = \text{seg}(C(i')) \) where \( C(i') \) is the \( i' \)-th child of \( C \). As \( i' < k \) would imply that some rectangles from \( R(j, C) \) are contained in \( A(j, C, k) \) and others are not, this contradicts the existence of the DP-cell \( (j, C, k, \{f_{C'}\}) \). So \( i' \geq k \). As \( i' = k \) implies that the DP-cell \( (j, C, k, \{f_{C'}\}) \) is a canonical DP-cell, this is also not possible. So \( i' \geq k + 1 \). So with substituting \( i = i' - 1 \) and using \( t' \geq 2 \) we know that the \( i' \)-th child of \( C \) exists and as \( \text{end}(C(i'-1)) = \text{beg}(C(i')) = s \) we get \( I = [\text{end}(C(i)), \text{end}(C)] \). \( \square \)

**Lemma 13. The DP-cell \( (j, C(k), 1, \{g_{C'}\}) \) exists.**

**Proof.** Since the values \( \{g_{C'}\} \) are taken from the values \( \{f_{C'}\} \) the upper bounds for them are automatically respected. Thus, it suffices to show that the last property of the definition of the DP-cells (see Section 3.1) holds. Let \( C' \) be a cell such that \( \{f_{C'}\} \neq \emptyset \). Suppose that at least one rectangle from \( R(j, C') \) intersects with \( A(j, C(k), 1) \). Then also at least one rectangle from \( R(j, C') \) intersects with \( A(j, C(k), 1) \) because \( A(j, C(k), 1) \subseteq A(j, C, k) \). As the DP-cell \( (j, C, k, \{f_{C'}\}) \) exists, every rectangle from \( R(j, C') \) is contained in \( A(j, C, k) \) and \( C' = C' = C ' \) is a descendant of \( C \). As \( I = [\text{end}(C(i)), \text{end}(C)] \) with \( i \geq k \) and \( \text{end}(C(k)) \neq \text{end}(C(k)) \) we obtain \( I \cap \text{end}(C(k)) = \emptyset \). This implies \( C' \neq C' \). So \( C' \) is a descendant of \( C \). Then \( C' \subseteq C' \) for a child \( C' \) of \( C \) if \( C' \neq C' \), then \( C' \subseteq C' \) or \( C' \subseteq \emptyset \) which contradicts that a rectangle in \( R(j, C') \) intersects with \( A(j, C(k), 1) \). So we get \( C' = C' \) and therefore \( C' = C' \) or \( C' = C' \). So the last property holds for \( j \) and each cell \( C' \).

Next, consider a job \( j' \neq j \) and a cell \( C' \) with \( R(j', C') \neq \emptyset \). Suppose again that at least one rectangle from \( R(j', C') \) intersects with \( A(j, C(k), 1) \). This implies \( j' \geq j \) and thus \( j' > j \). From the definition of the segments we know that there exists a cell \( C' \) such that the union of \( \text{Seg}(j', C') \) is contained in the union of \( \text{Seg}(C(k), C'') \) and additionally it holds that either \( C' = C' = C' \) is a descendant of \( C'' \). Then at least one rectangle from \( R(j', C') \) intersects with \( A(j, C(k), 1) \) and that \( C'' = C'' = C'' \) or \( C'' \subseteq \emptyset \). So the union of \( R(j', C') \) is contained in \( A(j, C(k), 1) \) and that \( C'' = C'' \) or \( C'' \subseteq \emptyset \). This proves the last property of the definition of our DP-cells and we conclude that the DP-cell \( (j, C(k), 1, \{g_{C'}\}) \) exists. \( \square \)

Now we consider the case \( k = K \) and we will consider the more complicated case \( k < K \) afterwards.

**Lemma 14. If \( k = K \) then the subproblems \( (j, C, k, \{f_{C'}\}) \) and \( (j, C(k), 1, \{g_{C'}\}) \) have the same set of solutions.**

**Proof.** By definition of \( A \) it holds that \( A(j, C, K) = A(j, C(K), 1) \), so \( R(j, C, K) = R(j, C(K), 1) \). Consider a ray \( L(I) \) for some interval \( I \). Then \( L(I) \subseteq A(j, C, K) \) is equivalent to \( L(I) \subseteq A(j, C(K), 1) \), so both sub-problems need to cover the same demands. It remains to verify that the values \( \{f_{C'}\} \) and \( \{g_{C'}\} \) yield equivalent conditions for the additional demands for some of the rays. The union of the intervals in \( C(C, K) \) is \( C(K) \) which is the same as the union of the intervals in \( C(C(K), 1) \). Let \( t \in C(k) \) with \( t \geq r_j \). Then there exists \( C'' \in C(C(K), 1) \) with \( t \in C'' \) and \( C'' \in C(C, K) \) with \( C'' \subseteq C' \). Note that \( f_{C'} = g_{C'}. \) This implies \( d([r_j, t]) + f_{C'} = d([r_j, t]) + g_{C''}, \) so the extended demands for each ray \( L([r_j, t]) \) with \( t \in C(k) \)

1342

Alexander Armbruster, Lars Rohwedder, and Andreas Wiese
are also the same. Altogether this shows that the sub-problems $(j, C, k, \{f_{C'}\})$ and $(j, C(k), 1, \{g_{C'}\})$ have the same set of solutions.

Since the subproblems $(j, k, \{f_{C'}\})$ and $(j, C(k), 1, \{g_{C'}\})$ have the same set of solutions, it follows that the optimal solutions are also the same. So this completes the proof for the case that $k = K$.

Now we consider the case that $k < K$.

**Lemma 15.** If $k < K$ then the DP-cell $(j, C, k + 1, \{f_{C'}\})$ exists.

**Proof.** Again we only need to check the last property of the definition of the DP-cells. Consider a job $j'$ and a cell $C'$. From the definition of the respective sets $A$ it follows that $A(j, C, k) = A(j, C(k), 1) \cup A(j, C(k), k + 1)$. A rectangle $R \in \mathcal{R}(j', C')$ is contained in $A(j, C(k), k + 1)$ if and only if it is contained in $A(j, C, k)$ and it does not intersect $A(j, C(k), 1)$. We already know that the DP-cells $(j, C, k, \{f_{C'}\})$ and $(j, C(k), 1, \{g_{C'}\})$ exist, so they fulfill the last property in the definition of a DP-cell. It follows that either

- all rectangles in $\mathcal{R}(j', C')$ are contained in $A(j, C, k)$ and none of them intersect $A(j, C(k), 1)$, or
- all rectangles in $\mathcal{R}(j', C')$ are contained in $A(j, C, k)$ and all rectangles in $\mathcal{R}(j', C')$ are contained in $A(j, C(k), 1)$, or
- no rectangle from $\mathcal{R}(j', C')$ intersects $A(j, C(k), 1)$.

In the first case every rectangle in $\mathcal{R}(j', C')$ is contained in $A(j, C, k)$ and in the second and third case no rectangle in $\mathcal{R}(j', C')$ intersects $A(j, C(k), 1)$. Thus, in all cases either all rectangles in $\mathcal{R}(j', C')$ are contained in $A(j, C, k + 1)$ or no rectangle in $\mathcal{R}(j', C')$ intersects $A(j, C, k + 1)$. If all rectangles in $\mathcal{R}(j', C')$ are contained in $A(j, C, k + 1)$ then all those rectangles are also contained in $A(j, C, k)$ which then implies that $C' = C$ or $C'$ is a descendant of $C$. This shows the last property of the definition of the DP-cells and therefore the DP-cell $(j, C, k + 1, \{f_{C'}\})$ exists.

**Lemma 16.** If $k < K$ then it holds that $\mathcal{R}' := \mathcal{R}' \cap \mathcal{R}_1(j, C(k), 1)$ is a feasible solution for $(j, C(k), 1, \{g_{C'}\})$.

**Proof.** It is immediate that $\mathcal{R}' \subseteq \mathcal{R}_1(j, C(k), 1)$. Consider a job $j'$ and a cell $C'$. Then $\mathcal{R}' \cap \mathcal{R}(j', C')$ forms a prefix of $\mathcal{R}(j', C')$ and as $\mathcal{R}(j, C(k), 1) \subseteq \mathcal{R}(j', C')$ either contains all rectangles in $\mathcal{R}(j', C')$ or none of them, also the $\mathcal{R}' \cap \mathcal{R}(j', C')$ forms a prefix of $\mathcal{R}(j', C')$. Now consider a ray $(l)(i) \in \mathcal{R}$ for an interval $I = [s, t]$ with $I \subseteq \mathcal{R}(j, C, k)$. If $j \notin C(k) \cup (i)$ then $(l)(i) \subseteq A(j, C, k)$ and $L(I)$ is covered by $\mathcal{R}$, otherwise $L(I) \subseteq A(j, C, k + 1)$ and $L(I)$ is covered by $\mathcal{R}_2$. In both cases $L(I)$ is covered by $\mathcal{R}_1 \cup \mathcal{R}_2$. Let $C' \subseteq A(j, C, k)$ and consider a ray $(l)(i)$ with $i \in C'$. First suppose that $C' \subseteq A(j, C(k), 1)$. Then $\mathcal{R}_1 \subseteq \mathcal{R}(j', C')$ and thus also $\mathcal{R}_1 \cap \mathcal{R}(j', C') \subseteq \mathcal{R}(j', C')$. Secondly, consider $C' \subseteq A(j, C, k) \setminus A(j, C(k), 1)$. Then $C' \subseteq A(j, C, k)$ and thus there exists $\mathcal{R}_1 \in C(j(k), 1)$ with $i \in C'$. Then $p(\mathcal{R}_1 \cap \mathcal{R}(j', C')) \subseteq d([r, t]) + g_{C'}$ and thus also $p(\mathcal{R}_1 \cap \mathcal{R}(j', C')) \subseteq d([r, t]) + g_{C'}$. Consequently, we have in both cases that $p(\mathcal{R}_1 \cap \mathcal{R}(j', C')) \subseteq d([r, t]) + f_{C'}$.

**3.5 Proof of Lemma 6**

Assume that $(j, c, k, \{f_{C'}\})$ is a non-canonical DP-cell and that $(j, c, k, \{f_{C'}\})$ it follows that $\mathcal{R}_1 \cap \mathcal{R}_2$ is also an optimal solution for $(j, c, k, \{f_{C'}\})$ which concludes the proof of Lemma 5.

**Lemma 19.** We have that $k < K$ and $\mathcal{R}_1(j, c, k) = \mathcal{R}_1(j, c, k + 1)$.

**Proof.** We first show that $\mathcal{R}(j, c)$ is non-empty and contained in $A(j, c, k)$. From the definition of $C(j, k)$ we conclude that the intervals in $C(j, k)$ are the intervals $[t, t+1]$ for $t \in A(j, c, k)$ and therefore $\mathcal{R}(j, c)$ contains at least one rectangle. It can only contain rectangles in sets $\mathcal{R}(j')$ with $j' \geq j$. As the jobs are ordered by their release times we know that $r_j \leq r_{j'}$ for all $j' \geq j$. So the leftmost $x$-coordinate of each rectangle belonging to some job $j' \geq j$ is at least $r_j$. Thus $A(j, c, k)$ can only contain rectangles whose leftmost $x$-coordinate is at least $r_j$. As $A(j, c, k)$ contains a rectangle, we obtain $r_j \leq \text{end}(C) - 1$. So $A(j, c, k)$ contains a rectangle in $\mathcal{R}(j)$.
a rectangle in \( R(j, C) \). Thus \( R(j, C) \) is non-empty. Due to the last property in the definition of the DP-cell \( (j, C, k, \{ f_{C'} \}) \) all rectangles in \( R(j, C, k) \) are contained in \( A(j, C, k) \).

The segments in \( \text{Seg}(j, C) \) are the intervals \([t, t + 1)\) with \( t \in \mathbb{N} \) and \( r_j \leq t \leq \text{end}(C) - 1 \). As all rectangles in \( R(j, C, k) \) are contained in \( A(j, C, k) \) we obtain \( \text{beg}(C) + k - 1 \leq r_j \). As \( \text{beg}(C) + k - 1 = r_j \) this yields the DP-cell \((j, C, k, \{ f_{\text{beg}(C)} \})\) is canonical which we assume to be not the case, this implies \( \text{beg}(C) + k \leq r_j \). We already showed \( r_j \leq \text{end}(C) - 1 \), thus \( \text{beg}(C) + k \leq \text{end}(C) - 1 \) and thus \( k \leq \text{end}(C) - \text{beg}(C) \). This implies that \( A(j, C, k + 1) \neq \emptyset \) and \( k \leq \text{end}(C) - \text{beg}(C) \leq K \).

We know that every rectangle contained in \( A(j, C, k) \) has a leftmost \( x \)-coordinate of at least \( r_j \). As \( \text{beg}(C) + k \leq r_j \) it is also contained in \( A(j, C, k + 1) \). This yields \( R(j, C, k) = R(j, C, k + 1) \). \( \square \)

Now we show the existence of the DP-cell \((j, C, k, \{ f_{C} \})\). Since the values \( \{ f_{C} \} \) are taken from the values \( \{ f_{C'} \} \), the upper bounds for them are automatically respected. Using Lemma 19 and the fact that the DP-cell \((j, C, k, \{ f_{C} \})\) exists, this implies the existence of the DP-cell \((j, C, k + 1, \{ f_{C} \})\).

In the remainder we show that the sub-problems \((j, C, k, \{ f_{C'} \})\) and \((j, C, k + 1, \{ f_{C} \})\) have the same set of solutions. We already proved \( R(j, C, k) = R(j, C, k + 1) \). So consider a ray \( L(I) \) for an interval \( I \) with \( L(I) \subseteq A(j, C, k) \). If \( L(I) \not\subseteq A(j, C, k) \backslash A(j, C, k + 1) \) then it does not intersect any rectangles so we do not need to check it by Lemma 3. And if \( L(I) \subseteq A(j, C, k + 1) \) then the demand for this ray is the same for both sub-problems. Consider an interval \( C' \subseteq C(C, k) \) and let \( t \in C' \) such that there is a ray \( L([r_j, t]) \). Then \( t \geq r_j \) and thus \( C' \subseteq C(C, k + 1) \). So the set of rays with additional demand \( \{ f_{C} \} \) etc. are also the same. Altogether, the sub-problems \((j, C, k, \{ f_{C} \})\) and \((j, C, k + 1, \{ f_{C} \})\) have indeed the same set of solutions. This also implies that they have the same optimal solutions.

REFERENCES

[1] Yossi Azar and Noam Nisan. 2008. Improved Online Algorithm for Weighted Flow Time. In Proceedings of FOCS. 427–437. https://doi.org/10.1109/FOCS.2008.00048
[2] Nikhil Bansal and Ho-Leung Chan. 2009. Weighted flow time does not admit \( O(1) \)-competitive algorithms. In Proceedings of SODA. 1238–1244. https://doi.org/10.5555/1490778.1490604
[3] Nikhil Bansal and Kedar Dhamdhere. 2007. Minimizing weighted flow time. ACM Transactions on Algorithms 3, 4 (2007), 39. https://doi.org/10.1145/1290672.1290676
[4] Nikhil Bansal and Kirk Pruhs. 2003. Server scheduling in the \( L_\infty \) norm: a rising tide lifts all boats. In Proceedings of STOC. 242–250. https://doi.org/10.1145/780542.780580
[5] Nikhil Bansal and Kirk Pruhs. 2004. Server Scheduling in the Weighted \( L_\infty \) Norm. In Proceedings of LATIN. 434–443. https://doi.org/10.1007/978-3-540-22626-7_47
[6] Nikhil Bansal and Kirk Pruhs. 2014. The Geometry of Scheduling. SIAM J. Comput. 43, 5 (2014), 1684–1698. https://doi.org/10.1137/130911317
[7] Jatin Batra, Naveen Garg, and Ankit Kumar. 2018. Constant Factor Approximation Algorithm for Weighted Flow Time on a Single Machine in Pseudo-Polynomial Time. In Proceedings of FOCS. 778–789. https://doi.org/10.1109/FOCS.2018.00070
[8] Michael A. Bender, S. Mukherji, and Rajmohan Rajaraman. 2004. Approximation Algorithms for Average Stretch Scheduling. Journal of Scheduling 7, 3 (2004), 195–222. https://doi.org/10.1023/B:JOSD.0000019681.52701.bf
[9] Peter Brucker, Jan Karel Lenstra, and AHG Rinnooy Kan. 1975. Complexity of machine scheduling problems. Stichting Mathematisch Centrum. Mathematische Besliskunde BW 43/75 (1975). https://doi.org/10.1016/S0167-5060(08)70743-X
[10] Chandra Chekuri and Sanjeev Khanna. 2002. Approximation schemes for preemptive weighted flow time. In Proceedings of STOC. 297–305. https://doi.org/10.1145/509907.509954
[11] Chandra Chekuri, Sanjeev Khanna, and An Zhu. 2001. Algorithms for minimizing weighted flow time. In Proceedings of STOC. 84–93. https://doi.org/10.1145/380752.380778
[12] Uriel Feige, Janardhan Kulkarni, and Shi Li. 2019. A polynomial time constant approximation for minimizing total weighted flow-time. In Proceedings of SODA. 1585–1595. https://doi.org/10.1137/1.9781611975348.96
[13] Lars Rohwedder and Andreas Wiese. 2021. \( (1 + \varepsilon) \)-approximation algorithm for preemptive weighted flow time on a single machine. In Proceedings of STOC. 1042–1055. https://doi.org/10.1145/3406235.3451075
[14] Andreas Wiese. 2018. Fixed-Parameter Approximation Schemes for Weighted Flowtime. In Proceedings of APPROX-RANDOM 28:1–28:19. https://doi.org/10.4230/LIPIcs APPROX-RANDOM.2018.28

Received 2022-11-07; accepted 2023-02-06