Constant Factor Approximation Algorithm for Weighted Flow Time on a Single Machine in Pseudo-polynomial time

Jatin Batra Naveen Garg Amit Kumar

Department of Computer Science and Engineering
IIT Delhi

Abstract

In the weighted flow-time problem on a single machine, we are given a set of n jobs, where each job has a processing requirement p_j, release date r_j and weight w_j. The goal is to find a preemptive schedule which minimizes the sum of weighted flow-time of jobs, where the flow-time of a job is the difference between its completion time and its released date. We give the first pseudo-polynomial time constant approximation algorithm for this problem. The running time of our algorithm is polynomial in n, the number of jobs, and P, which is the ratio of the largest to the smallest processing requirement of a job. Our algorithm relies on a novel reduction of this problem to a generalization of the multi-cut problem on trees, which we call `Demand Multi-cut` problem. Even though we do not give a constant factor approximation algorithm for the `Demand Multi-cut` problem on trees, we show that the specific instances of `Demand Multi-cut` obtained by reduction from weighted flow-time problem instances have more structure in them, and we are able to employ techniques based on dynamic programming. Our dynamic programming algorithm relies on showing that there are near optimal solutions which have nice smoothness properties, and we exploit these properties to reduce the size of DP table.

1 Introduction

Scheduling jobs to minimize the average waiting time is one of the most fundamental problems in scheduling theory with numerous applications. We consider setting where jobs arrive over time (i.e., have release dates), and need to be processed such that the average flow-time is minimized. The flow-time, F_j of a job j, is defined as the difference between its completion time, C_j, and release date, r_j. It is well known that for the case of single machine, the SRPT policy (Shortest Remaining Processing Time) gives an optimal algorithm for this objective.

In the weighted version of this problem, jobs have weights and we would like to minimize the weighted sum of flow-time of jobs. However, the problem of minimizing *weighted* flow-time (`WtdFlowTime`) turns out to be NP-hard and it has been widely conjectured that there should a constant factor approximation algorithm (or even PTAS) for it. In this paper, we make substantial progress towards this problem by giving the first constant factor approximation algorithm for this problem in pseudo-polynomial time. We show that the problem can be reduced to a generalization of the multi-cut problem on trees, which we call `Demand Multi-cut`. The `Demand Multi-cut` problem is a natural generalization of the multi-cut problem where edges have sizes and costs, and input paths (between terminal pairs) have demands. We would like to select a minimum cost subset of edges such that for every path in the input, the total size of edges in the path is at least the demand of the path. When all demands and sizes are 1, this is the usual multi-cut problem. The natural integer program for this problem has the property that all non-zero entries in any column of the constrain matrix are the same. Such integer programs, called *column restricted integer programs*, were studied by Chakrabarty et al. [3]. They showed that one can get a constant factor approximation algorithm for `Demand Multi-cut` provided one could prove that the integrality gap of the natural LP relaxations for the following two special cases is constant – (i) the version where the constraint matrix has 0-1 entries only, and (ii) the priority version, where paths and edges in the tree have priorities (instead of sizes and demands respectively), and we want to pick minimum cost subset of edges such that for each path, we pick at least one edge in it of priority which is at least the priority of this path. Although the first problem turns out to be easy, we do not know how to round the LP relaxation of the priority version. This is similar to the situation faced by Bansal and Pruhs [2], where they need to round the priority version of a geometric set cover problem. They appeal to the notion of union complexity [4] to get $O(\log\log P)$-approximation for this problem. It turns out the union complexity of the priority version of `Demand Multi-cut` is also unbounded (depends on the number of distinct priorities) [4], and so it is unlikely that this approach will yield a constant factor approximation.

However, it turns out that the specific instances of `Demand Multi-cut` incurred by our reduction have more structure, namely each node has at most 2 children, each path goes from an ancestor to a descendant, and the tree has $O(\log(nP))$ depth if we shortcut all degree 2 vertices. We show that one can effectively use dynamic programming techniques for such instances. We show that there is a near optimal solution which has nice "smoothness" properties so that the dynamic programming table can manage with storing small amount of information.

1.1 Related Work

There has been a lot of work on the `WtdFlowTime` problem on a single machine, though polynomial time constant factor approximation algorithm has remained elusive. Bansal and Dhamdhere [1] gave $O(\log W)$ approximation algorithm for this problem, where W is the ratio of the maximum to the minimum weight of a job. Chekuri et al. [6] gave an $O(\log^2 P)$-approximation algorithm, where P is the ratio of the largest to the smallest processing time of a job. Recently, Bansal and Pruhs [2] made significant progress towards this problem by giving an $O(\log \log P)$-approximation algorithm. In fact, their result applies to a more general setting where the objective function is $\sum_j f_j(C_j)$, where $f_j(C_j)$ is any monotone function of the completion time C_j of job j. Chekuri and Khanna [5] gave a quasi-PTAS for this problem, where the running time was $O(n^{O_\epsilon(\log W \log P)})$.

The multi-cut problem on trees is known to be NP-hard, and a 2-approximation algorithm was given by Garg et al. [7].

2 Preliminaries

An instance of the weighted flow-time problem is specified by a set of n jobs. Each job has a processing requirement p_j, weight w_j and release date r_j. We assume wlog that all of these quantities are integers, and let P denote the largest processing requirement of a job. We divide the time line into unit length *slots*– we shall often refer to the time slot $[t, t+1]$ as slot t. A feasible schedule needs to process a job j for p_j units after its release date. Note that we allow a job to be preempted. The weighted flow-time of a job is defined as $w_j \cdot (C_j - r_j)$, where C_j is the slot in which the job j finishes processing. The objective is to find a schedule which minimizes the sum over all jobs of their weighted flow-time.

Note that any schedule would occupy exactly $T = \sum_j p_j$ slots. We say that a schedule is *busy* if it does not leave any slot vacant even though there are jobs waiting to be finished. We can assume that the optimal schedule is a busy schedule (otherwise, we can always shift some processing back and improve the objective function). We also assume that any busy schedule fills the slots in $[0, T]$ (otherwise, we can break it into independent instances satisfying this property).

We shall also consider a generalization of the multi-cut problem on trees, which we call the `Demand Multi-cut` problem. Here, edges have cost and size, and demands are specified by ancestor-descendant paths. Each such path has a demand, and the goal is to select a minimum cost subset of edges such that for each path, the total size of selected edges in the path is at least the demand of this path.

In Section 2.1, we describe a well-known integer program for `WtdFlowTime`. This IP has variables $x_{j,t}$ for every job j, and time $t \geq r_j$, and it is supposed to be 1 if j completes processing after time t. The constraints in the IP consist of several covering constraints. However, there is an additional complicating factor that $x_{j,t} \leq x_{j,t-1}$ must hold for all $t \geq r_j$. To get around this problem, we propose a different IP in Section 3. In this IP, we define variables of the form $y(j, S)$, where S are exponentially increasing intervals starting from the release date of j. This variable indicates whether j is alive during the entire duration of S. The idea is that if the flow-time of j lies between 2^i and 2^{i+1}, we can count 2^{i+1} for it, and say that j is alive during the entire period $[r_j + 2^i, r_j + 2^{i+1}]$. Conversely, if the variable $y(j, S)$ is 1 for an interval of the form $[r_j + 2^i, r_j + 2^{i+1}]$, we can assume that it is also alive for previous intervals (because their total length would be at most 2^i). This allows us to decouple the $y(j, S)$ variables for different S. By an addition trick, we can ensure that these intervals are laminar for different jobs. From here, the reduction to the `Demand Multi-cut` problem is immediate (see Section 4 for details). In Section 5, we show that the specific instances of `Demand Multi-cut` obtained by such reductions have additional properties. We use the property that the tree obtained from shortcutting all degree two vertices is binary and has $O(\log(nP))$ depth. We shall use the term *segment* to define a maximal degree 2 (ancestor-descendant) path in the tree. So the property can be restated as – any root to leaf path has at most $O(\log(nP))$ segments. Any vertex in the tree needs to maintain the "state" of segments above it, where the state could mean the edges selected by the algorithm. This would require too much book-keeping. We use two ideas to reduce the size of this state – (i) We first show that the optimum can be assumed to have certain smoothness properties, which cuts down on the number of possible configurations. The smoothness property essentially says that the cost spent by the optimum on a segment does not vary by more than a constant factor as we go to neighbouring segments, (ii) If we could guess how much cost the optimal solution spends on a segment, and if it is possible to spend a similar amount on this segment and select edges of high capacity, then we could ignore the edges selected by the algorithm in higher segments (with respect to a vertex v and paths passing through it). If we are not able to do this, it should already give us some information about the edges selected by the optimum.

2.1 An integer program

We describe an integer program for the `WtdFlowTime` problem. This is well known (see e.g. [2]), but we give details for sake of completeness. We will have binary variables $x_{j,t}$ for every job j and time t, where $r_j \leq t \leq T$. This variable is meant to be 1 iff j is *alive* at time t, i.e., its completion time is at least t. Clearly, the objective function is $\sum_j \sum_{t \in [r_j,T]} w_j x_{j,t}$. We now specify the constraints of the integer program. Consider a time interval $I = [s,t]$, where $0 \leq s \leq t \leq T$, and s and t are integers. Let $l(I)$ denote the length of this time interval, i.e., $t - s$. Let $J(I)$ denote the set of jobs released during I, i.e., $\{j : r_j \in I\}$, and $p(J(I))$ denote the total processing time of jobs in $J(I)$. Clearly, the total volume occupied by jobs in $J(I)$ beyond I must be at least $p(J(I)) - l(I)$. Thus, we get the following integer program: (IP1)

$$\min \sum_j \sum_{t \in [r_j,T]} w_j x_{j,t} \tag{1}$$

$$\sum_{j \in J(I)} x_{j,t} p_j \geq p(J(I)) - l(I) \qquad \text{for all intervals } I = [s,t], 0 \leq s \leq t \leq T \tag{2}$$

$$x_{j,t} \leq x_{j,t-1} \qquad \text{for all jobs } j \text{, and time } t,\ r_j < t \leq T \tag{3}$$

$$x_{j,t} \in \{0,1\} \quad \text{for all } j,t$$

It is easy to see that this is a relaxation – given any schedule, the corresponding $x_{j,t}$ variables will satisfy the constraints mentioned above, and the objective function captures the total weighted flow-time of this schedule. The converse is also true – given any solution to the above integer program, there is a corresponding schedule of the same cost. We give the proof for sake of completeness. We first observe a simple property of a feasible solution to the integer program.

Claim 2.1. *Consider an interval $I = [s,t]$, $0 \leq s \leq t \leq T$. Let J' be a subset of $J(I)$ such that $p(J') > l(I)$. If x is a feasible solution to (IP1), then there must exist a job $j \in J'$ such that $x_{j,t} = 1$.*

Proof. Suppose not. Then the LHS of constraint (2) for I would be at most $p(J(I) \setminus J')$, whereas the RHS would be $p(J') + p(J(I) \setminus J') - l(I) > p(J(I) \setminus J')$, a contradiction. □

Theorem 2.2. *Suppose $x_{j,t}$ is a feasible solution to (IP1). Then, there is a schedule for which the total weighted flow-time is equal to the cost of the solution $x_{j,t}$.*

Proof. We show how to build such a schedule. The integral solution x gives us deadlines for each job. For a job j, define d_j as one plus the last time t such that $x_{j,t} = 1$. Note that $x_{j,t} = 1$ for every $t \in [r_j, d_j)$. We would like to find a schedule which completes each job by time d_j : if such a schedule exists, then the weighted flow-time of a job j will be at most $\sum_{t \geq r_j} w_j x_{j,t}$, which is what we want.

It is natural to use the Earliest Deadline First rule. We build the schedule from time $t = 0$ onwards. At any time t, we say that a job j is *alive* if $r_j \leq t$, and j has not been completely processed by time t. Starting from time $t = 0$, we process the alive job with earliest deadline d_j during $[t, t+1]$. We need to show that every job will complete before its deadline. Suppose not. Let j be the job with the earliest deadline which is not able to finish by d_j. Let t be first time before d_j such that the algorithm processes a job whose deadline is more than d_j during $[t-1, t]$, or it is idle during this time slot (if there is no such time slot, it must have busy from time 0 onwards, and so set t to 0). The algorithm processes jobs whose deadline is at most d_j during $[t, d_j]$ – call these jobs J'. We also claim that jobs in J' were released after t – indeed if such a job was released before time t, it would have been alive at time $t-1$ (since it gets processed after time t). Further its deadline is at most d_j, and so, the algorithm should not be processing a job whose deadline is more than d_j during $[t-1, t]$ (or being idle). But now, consider the interval $I = [t, d_j]$. Observe that $l(I) < p(J')$ – indeed, $j \in J'$ and it is not completely processed during I, but the algorithm processes jobs from J' only during I. Claim 2.1 now implies that there must be a job j' in J' for which $x_{j',d_j} = 1$ – but then the deadline of j' is more than d_j, a contradiction. □

3 A Different Integer Program

We now write a weaker integer program, but it has more structure in it. We first assume that T is a power of 2 – if not, we can pad the instance with a job of zero weight (this will increase the ratio P by at most a factor n only). Let T be 2^ℓ. We now divide the time line into nested dyadic segments. A dyadic segment is an interval of the form $[i \cdot 2^s, (i+1) \cdot 2^s]$ for some non-negative integers i and s (we shall use segments to denote such intervals to avoid any confusion with intervals used in the integer program). For $s = 0, \ldots, \ell$, we define $\mathcal{S}_s$ as the set of dyadic segments of length 2^s starting from 0, i.e., $\{[0, 2^s], [2^s, 2 \cdot 2^s], \ldots, [i \cdot 2^s, (i+1) \cdot 2^s], \ldots, [T - 2^s, T]\}$. Clearly, any segment of $\mathcal{S}_s$ is contained inside a unique segment of $\mathcal{S}_{s+1}$. Now, for every job j we shall define a sequence of dyadic segments $\texttt{Seg}(j)$. The sequence of segments in $\texttt{Seg}(j)$ partition the interval $[r_j, T]$. The construction of $\texttt{Seg}(j)$ is described in Figure 1 (also see the example in Figure 2). It is easy to show by induction on s that the parameter t at the beginning of iteration s in Step 2 of the algorithm is a multiple of 2^s. Therefore, the segments added during the iteration for s belong to $\mathcal{S}_s$. Although we do not specify for how long we run the for loop in Step 2, we stop when t reaches T (this will always happen because t takes values from the set of end-points in the segments in $\cup_s \mathcal{S}_s$). Therefore the set of segments in $\texttt{Seg}(j)$ are disjoint and cover $[r_j, T]$.

For a job j and segment $S \in \texttt{Seg}(j)$, we shall refer to the tuple (j, S) as a *job-segment*. For a time t, we say that $t \in (j, S)$ (or (j, S) contains t) if $[t, t+1] \subseteq S$. We now show a crucial nesting property of these segments.

Lemma 3.1. *Suppose (j, S) and (j', S') are two job-segments such that there is a time t for which $t \in (j, S)$ and $t \in (j', S')$. Suppose $r_j \leq r_{j'}$, and $S \in \mathcal{S}_s, S \in \mathcal{S}_{s'}$. Then $s \geq s'$.*

Proof. We prove this by induction on t. When $t = r_{j'}$, this is trivially true because s' would be 0. Suppose it is true for some $t \geq r_{j'}$. Let (j, S) and (j', S') be the job segments containing t. Suppose $S \in \mathcal{S}_s, S' \in S_{s'}$. By induction hypothesis, we know that $s \geq s'$. Let $(j', \tilde{S}')$ be the job-segment containing $t+1$, and let $\tilde{S}' \in \mathcal{S}_{\tilde{s}'}$ (S' could be same as $\tilde{S}'$). We know that $\tilde{s}' \leq s'+1$. Therefore,

Algorithm FormSegments(j)

1. Initialize $t \leftarrow r_j$.
2. For $s = 0, 1, 2, \ldots,$
   - (i) If t is a multiple of 2^{s+1},
     - add the segments (from the set $\mathcal{S}_s$) $[t, t+2^s], [t+2^s, t+2^{s+1}]$ to $\texttt{Seg}(j)$
     - update $t \leftarrow t + 2^{s+1}$.
   - (ii) Else add the segment (from the set $\mathcal{S}_s$) $[t, , t+2^s]$ to $\texttt{Seg}(j)$.
     - update $t \leftarrow t + 2^s$.

Figure 1: Forming $\texttt{Seg}(j)$.

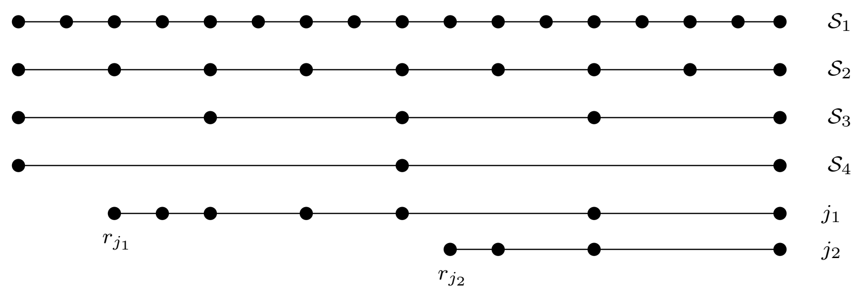

Figure 2: The dyadic segments $\mathcal{S}_1, \ldots, \mathcal{S}_4$ and the corresponding $\texttt{Seg}(j_1), \texttt{Seg}(j_2)$ for two jobs j_1, j_2

the only interesting case is $s = s'$ and $\tilde{s}' = s' + 1$. Since $s = s'$, the two segments S and S' must be same (because all segments in $\mathcal{S}_s$ are mutually disjoint). Since $t \in S, t+1 \notin S$, it must be that $S = [l, t+1]$ for some l. The algorithm for constructing $\texttt{Seg}(j')$ adds a segment from $\mathcal{S}_{s'+1}$ after adding S' to $\texttt{Seg}(j')$. Therefore $t+1$ must be a multiple of $2^{s'+1}$. What does the algorithm for constructing $\texttt{Seg}(j)$ do after adding S to $\texttt{Seg}(j)$? If it adds a segment from $\mathcal{S}_{s+1}$, then we are done again. Suppose it adds a segment from $\mathcal{S}_s$. The right end-point of this segment would be $(t+1)+2^s$. After adding this segment, the algorithm would add a segment from $\mathcal{S}_{s+1}$ (as it cannot add more than 2 segments from $\mathcal{S}_s$ to $\texttt{Seg}(j)$). But this can only happen if $(t+1)+2^s$ is a multiple of 2^{s+1} – this is not true because $(t+1)$ is a multiple of 2^{s+1}. Thus we get a contradiction, and so the next segment (after S) in $\texttt{Seg}(j)$ must come from $\mathcal{S}_{s+1}$ as well. □

We now write a new IP. The idea is that if a job j is alive at some time t, then we will keep it alive during the entire duration of the segment in $\texttt{Seg}(j)$ containing t. Since the segments in $\texttt{Seg}(j)$ have lengths in exponentially increasing order (except for two consecutive segments), this will not increase the weighted flow-time by more than a constant factor. For each job segment (j, S) we have a binary variable $y(j, S)$, which is meant to be 1 iff the job j is alive during the entire duration S. For each job segment (j, S), define its weight $w(j, S)$ as $w_j \cdot l(S)$ – this is the contribution towards weighted flow-time of j if j remains alive during the entire segment S. We get the following integer program (IP2):

$$\min \sum_j \sum_s w(j,S)y(j,S) \tag{4}$$

$$\sum_{(j,S):j\in J(I),t\in(j,S)} y(j,S)p_j \geq p(J(I)) - l(I) \qquad \text{for all intervals } I=[s,t], 0\leq s\leq t\leq T \tag{5}$$

$$y(j,S)\in\{0,1\} \quad \text{for all job segments } (j,S)$$

Observe that for any interval I, the constraint (5) for I has precisely one job segment for every job which gets released in I. Another interesting feature of this IP is that we do not have constraints corresponding to (3), and so it is possible that $y(j,S)=1$ and $y(j,S')=0$ for two job segments (j,S) and (j,S') even though S' appears before S in $\texttt{Seg}(j)$. We now relate the two integer programs.

Lemma 3.2. *Given a solution x for (IP1), we can construct a solution for (IP2) of cost at most 8 times the cost of x. Similarly, given a solution y for (IP2), we can construct a solution for (IP1) of cost at most 4 times the cost of y.*

Proof. Suppose we are given a solution x for (IP1). For every job j, let d_j be the highest t for which $x_{jt}=1$. Let the segments in $\texttt{Seg}(j)$ (in the order they were added) be $S_1,S_2,\ldots$. Let S_{i_j} be the segment in $\texttt{Seg}(j)$ which contains d_j. Then we set $y(j,S_i)$ to 1 for all $i\leq i_j$, and $y(j,S_i)$ to 0 for all $i>i_j$. This defines the solution y. First we observe that y is feasible for (IP2). Indeed, consider an interval $I=[s,t]$. If $x_{jt}=1$ and $j\in J(I)$, then we do have $y(j,S)=1$ for the job segment (j,S) containing t. Therefore, the LHS of constraints (2) and (5) for I are same. Also, observe that

$$\sum_{S\in\texttt{Seg}(j)} y(j,S)w(j,S) = \sum_{i=1}^{i_j} w_j\cdot l(S_i) \leq w_j 4l(S_{i_j}),$$

where the last inequality follows from the fact that there are at most two segments from any particular set $\mathcal{S}_s$ in $\texttt{Seg}(j)$, and so, the length of every alternate segments in $\texttt{Seg}(j)$ increase exponentially. So, $\sum_{i=1}^{i_j} l(s_i) \leq 2\left(l(S_{i_j})+l(S_{i_j-2})+l(S_{i_j-4})+\cdots\right) \leq 4\cdot l(S_{i_j})$. Finally observe that $l(S_{i_j})\leq 2(d_j-r_j)$. Indeed, the length of $S_{i_{j-1}}$ is at least half of that of S_{i_j}. So,

$$l(S_{i_j}) \leq 2l(S_{i_j-1}) \leq 2(d_j-r_j).$$

Thus, the total contribution to the cost of y from job segments corresponding to j is at most $8w_j(d_j-r_j) = 8w_j\sum_{t\geq r_j} x_{j,t}$. This proves the first statement in the lemma.

Now we prove the second statement. Let y be a solution to (IP2). For each job j, let S_{i_j} be the last job segment in $\texttt{Seg}(j)=\{S_1,S_2,\ldots\}$ for which $y(j,S)$ is 1. We set $x_{j,t}$ to 1 for every $t\leq d_j$, where d_j is the right end-point of S_{i_j}, and 0 for $t>d_j$. It is again easy to check that x is a feasible solution to (IP1). For a job j the contribution of j towards the cost of x is

$$w_j(d_j-r_j) = w_j\cdot\sum_{i=1}^{i_j} l(S_i) \leq 4w_j\cdot l(S_{i_j}) \leq 4\cdot \sum_{(j,S)\in\texttt{Seg}(j)} w(j,S)y(j,S).$$

□

The above lemma states that it is sufficient to find a solution for (IP2). Note that (IP2) is a covering problem. It is also worth noting that the constraints (5) need to be written only when a job segment starts or ends, and therefore (IP2) can be turned into a polynomial size integer program.

4 Reduction to `Demand Multi-cut` on Trees

We now show that (IP2) can be viewed as a covering problem on trees. We first put a tree structure on the set of job segments. This is possible because of the property stated in Lemma 3.1. Order the jobs according to release dates (breaking ties arbitrarily) – let $\prec_J$ be this total ordering (so, $j \prec_J j'$ implies that $r_j \leq r_{j'}$).

Construct a directed graph $\mathcal{T} = (V, E)$ as follows. The vertex set V is the set of all job segments (j, S). Every vertex (j, S) will have indegree 1 (except for the ones corresponding to the first job in the ordering $\prec_J$). For such a vertex (j, S), let j' be the job immediately preceding j in the total order $\prec_J$. Since the job segments in $\texttt{Seg}(j')$ partition $[r_{j'}, T]$, and $r_{j'} \leq r_j$, there is a pair (j', S') in $\texttt{Seg}(j')$ such that S' intersects S, and so contains S, by Lemma 3.1. We add an edge from (j', S') to (j, S). Notice that $\mathcal{T}$ does not have any cycles because $((j', S'), (j, S))$ is an arc only if $j' \prec_J j$. Since every vertex has indegree at most 1, it follows that each connected component of $\mathcal{T}$ (in the undirected sense) is a rooted tree, with edges pointing aways from the root. We will think of the graph as a forest where each component is an undirected rooted tree (since the edges are directed away from the root, we need not maintain this information).

Fix an interval $I = [s, t]$ and consider the constraint (5) corresponding to it. Let P be the vertices in $\mathcal{T}$ corresponding to the job segments appearing in the LHS of this constraint.

Lemma 4.1. *The vertices in P form a path in $\mathcal{T}$ from an ancestor to a descendant.*

Proof. Let $j_1, \ldots, j_k$ be the jobs which are released in I arranged according to $\prec_J$. Note that these will form a consecutive subsequence of the sequence obtained by arranging jobs according to $\prec_J$. Each of these jobs will have exactly one job segment (j_i, S_i) appearing on the LHS of this constraint (because for any such job j_i, the segments in $\texttt{Seg}(j_i)$ partition $[r_{j_i}, T]$). All these job segments contain t, and so, these segment intersect. Now, by construction of $\mathcal{T}$, it follows that the parent of (j_i, S_i) in the tree $\mathcal{T}$ would be (j_{i-1}, S_{i-1}). This proves the claim. □

Thus, the problem considered by (IP2) can be equivalently stated as follows. We are given a rooted tree $\mathcal{T} = (V, E)$ [1]. For every vertex v, we have a cost c_v and size p_v. We are also given a set of ancestor-descendant paths $\mathcal{P}$ in $\mathcal{T}$. For every path $P \in \mathcal{P}$, we are given a demand $d(P)$. Our goal is to pick a minimum cost subset of vertices V' such that for every path $P \in \mathcal{P}$, the set of vertices in $V' \cap P$ have total size at least $d(P)$. Note that this is equivalent to the problem considered by (IP2) because a vertex corresponding to the job segment (j, S) has cost $w(j, S)$ and size p_j. Similarly a path P corresponding to the constraint (5) for an interval I has demand $p(J(I)) - l(I)$. Note that in the reduction above, size of V in $\mathcal{T}$ is at most $O(n \log P)$, where n is the number of jobs, and so this is a polynomial time reduction.

We modify the tree covering problem slightly as follows. we will assume that edges, instead of vertices, have size and cost. This is wlog because we can transform an instance of the problem as

[1] Earlier every component of $\mathcal{T}$ was a rooted tree. But we can treat each component as an independent problem.

follows. For each non-root vertex v, let e be the parent edge of v in $\mathcal{T}$. We define cost c_e and size p_e of e as c_v and p_v respectively (note that now v does not have any size or profit). For the root, we add a new edge above it, and define its p_e, c_e values to be those of the root vertex. Similarly, if P was a path from v to w (v being the ancestor of w) in $\mathcal{P}$, then we replace it by a path which starts from the parent of v and ends at w (note that this argument requires that all paths in $\mathcal{P}$ are ancestor-descendant paths). We call this problem `Demand Multi-cut` problem.

5 Approximation Algorithm for the `Demand Multi-cut` problem

In this section we give a constant factor approximation algorithm for the special class of `Demand Multi-cut` problems which arise in the reduction from `WtdFlowTime`. We first consider the special case when $\mathcal{T}$ is a line in Section 5.1. Subsequently we consider the general case in Section 5.2.

5.1 Special Case of Line Graph

Assume that $\mathcal{T} = (V, E)$ is a line graph with edges $e_1, \ldots, e_m$ as we go from left to right. The following integer program (IP3) captures the `Demand Multi-cut` problem for such instances:

$$\min \sum_{v \in E} c_e x_e \tag{6}$$

$$\sum_{e \in P} p_e x_e \geq d(P) \qquad \text{for all paths } P \in \mathcal{P} \tag{7}$$

$$x_e \in \{0, 1\} \quad \text{for all } e \in E \tag{8}$$

Such an IP comes under the class of Column Restricted Covering IP as described in [3]. Chakrabarty et al. [3] show that one can obtain a constant factor approximation algorithm for this problem provided one can prove that the integrality gaps of the LP relaxations for the following two special class of problems are constant: (i) 0-1 instances where the p_e values are either 0 or 1, (ii) priority versions where paths in $\mathcal{P}$ and edges have priorities (which can be thought of as positive integers), and the selected edges satisfy the property that for each path $P \in \mathcal{P}$, we have selected at least one edge of priority at least that of $\mathcal{P}$ (it is easy to check that this is a special case of `Demand Multi-cut` problem by assigning exponentially increasing demands to paths of increasing priority, and similarly for edges).

Consider the class of 0-1 instances first. We need to consider only those edges for which p_e is 1 (contract the one for which p_e is 0). Now observe that the constraint matrix on the LHS in (IP3) has consecutive ones property (order the paths in $\mathcal{P}$ in increasing order of left end-point and write the constraints in this order). Therefore, the relaxed LP has integrality gap of 1.

Rounding the Priority Version We now consider the priority version of this problem. For each edge $e \in E$, we now have an associated priority p_e (instead of size), and each path in $\mathcal{P}$ also has a priority demand $p(P)$, instead of its demand. We shall use the notion of *shallow cell complexity* used in [4]. We need to argue about the following LP relaxation:

$$\min \sum_{e \in E} c_e x_e \tag{9}$$

$$\sum_{e \in P : p_e \geq p(P)} x_e \geq 1 \qquad \text{for all paths } P \in \mathcal{P} \tag{10}$$

$$x_e \geq 0 \quad \text{for all } e \in E \tag{11}$$

Let A be the constraint matrix on the LHS above. We first notice the following property of A.

Claim 5.1. *Let $A^\star$ be a subset of columns of A obtained by selecting a subset of n columns. For a parameter $k, 0 \leq k \leq n$, there are at most $k^2 n$ distinct rows in $A^\star$ with k or fewer 1's (two rows of $A^\star$ are distinct if they are not same as row vectors).*

Proof. Columns of A correspond to edges. Contract all edges which are not in $A^\star$. Let $E^\star$ be the edges in the resulting graph. Each path in $\mathcal{P}$ now maps to a new path obtained by contracting these edges. Let $\mathcal{P}^\star$ denote the set of resulting paths. For a path $P \in \mathcal{P}^\star$, let $E(P)$ be the edges in P whose priority is at least that of P. In the constraint matrix $A^\star$, the constraint for a path P has 1's in exactly the edges in $E(P)$. We can assume that the set $E(P)$ is distinct for every path $P \in \mathcal{P}^\star$ (because we are interested in counting the number of paths with distinct sets $E(P)$).

Let $\mathcal{P}^\star(k)$ be the paths in $\mathcal{P}^\star$ for which $|E(P)| \leq k$. We need to count the cardinality of this set. Fix an edge $e \in E^\star$, let $E^\star(e)$ be the edges in $E^\star$ of priority at least that of e. Let P be a path in $\mathcal{P}^\star(k)$ which has e as the least priority edge in $E(P)$ (breaking ties arbitrarily). Let e_l and e_r be the leftmost and the rightmost edges in $E(P)$ respectively. Note that $E(P)$ is exactly the edges in $E^\star(e)$ which lie between e_l and e_r. Since there are at most k choices for e_l and e_r (look at the k edges to the left and to the right of e in the set $E^\star(e)$), it follows that there are at most k^2 paths P in $\mathcal{P}^\star(k)$ which have e as the least priority edge in $E(P)$. For every path in $\mathcal{P}^\star(k)$, there are at most $|E^\star| = n$ choices for the least priority edge. Therefore the size of $\mathcal{P}^\star(k)$ is at most nk^2. □

It now follows from Theorem 1.1 in [4] that the integrality gap of the LP relaxation of (IP3) is $O(1)$.

5.2 The Dynamic Programming Algorithm

We now describe the dynamic program (DP) for `Demand Multi-cut` on trees. We first show that the input instances obtained by reduction from `WtdFlowTime` have more structure in them. Subsequently, we show that an optimal solution can be modified to have more structure in it (at the expense of constant loss in cost). Finally, we show that the DP table needs to maintain small amount of information at each of its table entries, and so, these entries can be computed in pseudo-polynomial time.

5.2.1 Structure of Input Instances

We shall use the following properties of an instance of `Demand Multi-cut` on trees.

Lemma 5.2. *Let $\mathcal{I}$ be an instance of* `Demand Multi-cut` *obtained by reduction from an instance $\mathcal{I}'$ of* `WtdFlowTime`*. Let $\mathcal{T} = (V, E)$ be the tree in $\mathcal{I}$. Then we can assume wlog that $\mathcal{T}$ satisfies the following properties (at the expense of constant loss in the cost of the optimal solution):*

- *(Binary Tree) Every non-leaf node in $\mathcal{T}$ has either 1 or 2 children.*
- *(Low Depth) Let $\mathcal{T}'$ be the tree obtained from $\mathcal{T}$ by shortcutting all degree 2 nodes. Then $\mathcal{T}'$ has height $O(\log(nP))$.*

Proof. Recall that each vertex v in $\mathcal{T}$ corresponds to a dyadic interval S_v, and if w is a child of v then S_w is contained in S_v. Now, consider a vertex v with S_v of size 2^s and suppose it has more than 2 children. Since the dyadic intervals for the children are mutually disjoint and contained in S_v, each of these will be of size at most 2^{s-1}. Let S_v^1 and S_v^2 be the two dyadic intervals of length 2^{s-1} contained in S_v. Consider S_v^1. Let $w_1, \ldots, w_k$ be the children of v for which the corresponding interval is contained in S_v^1. If $k > 1$, we create a new node w below v (with corresponding interval being S_v^1) and make $w_1, \ldots, w_k$ children of v. The cost and size of the edge (v, w) is 0. We proceed similarly for S_v^2. Thus, each node will now have at most 2 children. Note that we will blow up the number of vertices by a factor 2 only.

Let $p_{\max}$ and $p_{\min}$ denote the maximum and the minimum size of a job in the instance $\mathcal{I}'$. In the instance $\mathcal{I}$, we will include all edges for which the corresponding job intervals have length at most $p_{\min}$. For a job j, the total cost of such job intervals would be at most $4w_jp_j$ (as in the proof of Lemma 3.2). Note that the cost of any optimal solution for $\mathcal{I}'$ is at least $\sum_j w_jp_j$, and so we are incurring an extra cost of at most 4 times the cost of the optimal solution. Now, we can show the second property. Consider a root to leaf path in $\mathcal{T}'$, and let $v_1, \ldots, v_k$ be the vertices in this path. Since each v_i has two children in $\mathcal{T}$, the dyadic interval corresponding to v_{i+1} will have length at most half of that for v_i. Since the length of the dyadic interval corresponding to the root has length at most $T \leq np_{\max}$, and that for the leaf has length at least $p_{\min}$, it follows that k has to be $O(\log(nP))$. □

For rest of the discussion, we fix an instance $\mathcal{I}$ of `Demand Multi-cut` satisfying the above two properties. We shall use $\mathcal{T} = (V, E)$ to denote the tree in $\mathcal{I}$ and $\mathcal{P}$ to denote the ancestor-descendant paths in $\mathcal{I}$. Each path P in $\mathcal{P}$ has a demand $d(P)$, and edges e in E have cost c_e and size s_e. The density of an edge e is the ratio c_e/s_e. We shall use $\mathcal{T}'$ to denote the tree obtained from $\mathcal{T}$ by shortcutting all degree 2 nodes. Each edge in $\mathcal{T}'$ corresponds to a a path in $\mathcal{T}$ between two nodes in the vertex set $V(\mathcal{T}')$ of $\mathcal{T}'$ such that all internal nodes have degree 2. We shall denote such paths in $\mathcal{T}$ as *segments*.

We shall also assume the following properties of $\mathcal{I}$, which result in at most constant loss in cost of the optimal solution:

(P1) The costs of edges are polynomially bounded: We can guess the maximum cost of an edge selected by the optimal solution. If this cost is c, then we can select all edges of cost at most c_e/n^2, where n is the number of edges. This will increase the cost of an optimal solution by $(1 + 1/n)$ factor only.

(P2) Each edge has density which is an integral power of 16: we can round the costs up to the nearest power of 16, this will affect the cost of our algorithm by a constant factor only. We say that an edge e has *density class* ρ if its density is 16^ρ. Note that density class of an edge is an integer and lies in the range $[\Delta_{\min}, \Delta_{\max}]$, where $\Delta_{\max} - \Delta_{\min}$ is $O(\log(nP))$. This is because property (P1) implies that costs are polynomially bounded, and sizes of edges lie in the range $[p_{\min}, p_{\max}]$. [2]

[2] We added some edges of cost 0 in the proof of Lemma 5.2. We can assume that they are always selected, and so, their density class will not matter.

(P3) Each path in $\mathcal{P}$ contains at least one vertex in $V(\mathcal{T}')$: For each segment s in $\mathcal{T}$, let $\mathcal{P}(s)$ be the paths which are confined within this segment. For each s, we have an independent instance of line graph corresponding to $\mathcal{P}(s)$, and we can get a constant approximation for such instances using the algorithm in Section 5.1. The total cost of edges selected by solving each of these instances independently is at most a constant times the cost of the optimal solution for $\mathcal{I}$. We can now ignore these paths, and solve the remaining instance (at the expense of doubling the cost of the algorithm).

Henceforth, we assume that the instance $\mathcal{I}$ satisfies the properties in Lemma 5.2 and (P1)-(P3).

5.2.2 Properties of a near-optimal solution

Let $\mathcal{A}$ be an optimal solution to $\mathcal{I}$. We give some notations first. Let $S(\mathcal{T})$ denote the set of segments in $\mathcal{T}$. We shall use the term *cell* to denote a pair (s, ρ) of segment and density class, where $s \in S(\mathcal{T}), \rho \in [\Delta_{\min}, \Delta_{\max}]$. For a cell (s, ρ), let $E(s, \rho)$ be the edges of density class ρ in the segment s. Let $E^{\mathcal{A}}(s, \rho)$ denote the subset of $E(s, \rho)$ which is selected by $\mathcal{A}$. Let $c^{\mathcal{A}}(s, \rho)$ denote the total cost of the edges in $E^{\mathcal{A}}(s, \rho)$. We first modify $\mathcal{A}$ to a solution $\bar{\mathcal{A}}$ which incurs slightly more cost that $E^{\mathcal{A}}(s, \rho)$ on each cell (s, ρ), but has nice *smoothness* properties. The smoothness properties essentially says that the quantity $E^{\bar{\mathcal{A}}}(s, \rho)$ will vary slowly as we change ρ or s (i.e., move away from s to a nearby segment).

For two segments s_1 and s_2 in $S(\mathcal{T})$, define the distance between them to be the number of vertices of $V(\mathcal{T}')$ which lie on the unique path in $\mathcal{T}$ joining these two segments (s_1 need not be ancestor or descendant of s_2). For a segment s, let $N_i(s)$ be the segments at distance i from it. These are exactly the edges which are distance i away from the edge corresponding to s in the tree $\mathcal{T}'$. Since $\mathcal{T}'$ is a binary tree (Lemma 5.2), it follows that $|N_i(s)| \leq 2^{i+1}$. We say that a segment s is a *parent* (or ancestor) of segment s' if the corresponding edges have this relation in the tree $\mathcal{T}'$.

Lemma 5.3. *We can find values $\bar{c}(s, \rho)$ for each cell $(s, \rho) \in S(\mathcal{T})$ such that the following properties are satisfied: (i) for every cell (s, ρ), $\bar{c}(s, \rho)$ is a power of 2, and $\bar{c}(s, \rho) \geq c^{\mathcal{A}}(s, \rho)$, (ii)$\sum_{(s,\rho) \in S(\mathcal{T})} \bar{c}(s, \rho) \leq 16 \cdot \sum_{(s,\rho) \in S(\mathcal{T})} c^{\mathcal{A}}(s, \rho)$ and (iii) (smoothness) for every pair of segments s, s', where s' is the parent of s, and density class ρ,*

$$8\bar{c}(s, \rho+1) \geq \bar{c}(s, \rho) \geq \bar{c}(s, \rho+1)/8, \text{ and } 8\bar{c}(s', \rho) \geq \bar{c}(s, \rho) \geq \bar{c}(s', \rho)/8.$$

Proof. We define

$$\bar{c}(s, \rho) := \sum_{i \geq 0} \sum_{s' \in N_i(s)} \sum_{j} \frac{c^{\mathcal{A}}(s', \rho + i + j)}{4^{i+|j|}},$$

where i varies over non-negative integers, j varies over integers and the range of i, j are such that $\rho + i + j$ remains a valid density class. Note that $c(s, \rho)$ is a not a power of 2 yet, but we will round it up later. As of now, $c(s, \rho) \geq c^{\mathcal{A}}(s, \rho)$ because the term on RHS for $i = 0, j = 0$ is exactly $c^{\mathcal{A}}(s, \rho)$.

Let us check the total sum of these values. When we add $c(s, \rho)$ for all the cells (s, ρ), let us count the total contribution towards terms containing $c^{\mathcal{A}}(s', \rho')$ on the RHS. For every segment $s \in N_i(s')$, and density class $\rho' + j$, it will receive a contribution of $\frac{1}{2^{i+|j|}}$. Since $|N_i(s')| \leq 2^i + 1$, this is at most

$$\sum_{i \geq 0} \sum_{j} \frac{2^{i+1}}{4^{i+|j|}} \leq \sum_{i \geq 0} \frac{2^{i+2}}{4^i} \leq 8.$$

Now consider the third condition. Consider the expressions for $\bar{c}(s,\rho)$ and $\bar{c}(s',\rho)$ where s' is the parent of s. If a segment if at distance i from s, its distance from s' is either i or $i \pm 1$. Therefore, the coefficients of $c^{\mathcal{A}}(s'',\rho'')$ in the expressions for $\bar{c}(s,\rho)$ and $\bar{c}(s',\rho)$ will differ by a factor of at most 4. The same observation holds for $\bar{c}(s,\rho)$ and $\bar{c}(s,\rho+1)$. It follows that

$$4\bar{c}(s,\rho+1) \geq \bar{c}(s,\rho) \geq \bar{c}(s,\rho+1)/4, \text{ and } 4\bar{c}(s',\rho) \geq \bar{c}(s,\rho) \geq \bar{c}(s',\rho)/4.$$

Finally, we round all the $\bar{c}(s,\rho)$ values up to the nearest power of 2. We will lose an extra factor of 2 in the statements (ii) and (iii) above. $\square$

Modifying $\mathcal{A}$ to $\bar{\mathcal{A}}$ We now define a new algorithm $\bar{\mathcal{A}}$ which is "allowed" a budget of $\bar{c}(s,\rho)$ as in the statement of Lemma 5.3 for selecting edges from $E(s,\rho)$. Note that it may not be able to spend all this budget because (i) the total cost of edges in $E(s,\rho)$ may be smaller than this budget, or (ii) there are edges in $E(s,\rho)$ which are more expensive than this budget. We also need to specify which edges in $E(s,\rho)$ are being selected by $\bar{\mathcal{A}}$. We claim that this can be done in a canonical way. For each cell (s,ρ), we use the Algorithm $\mathcal{B}$ in Figure 3 to figure out the edges in $E(s,\rho)$, which get selected by $\bar{\mathcal{A}}$. We will denote these edges by $E^{\bar{\mathcal{A}}}(s,\rho)$. Essentially $\bar{\mathcal{A}}$ starts from both end-points of s and keeps on picking edges in $E(s,\rho)$ till it exhausts (constant times) its budget. We now show that $\bar{\mathcal{A}}$ is indeed feasible and is not much more expensive than $\mathcal{A}$.

Algorithm $\mathcal{B}$:

Input: A cell (s,ρ) and budget $\bar{c}(s,\rho)$.
1. Initialize $E^{\bar{\mathcal{A}}}(s,\rho)$ to emptyset.
2. Let $E'(s,\rho)$ be the edges in $E(s,\rho)$ whose cost is at most $\bar{c}(s,\rho)$.
3. Starting from the top end-point of s, keep on selecting edges of $E'(s,\rho)$ in the set $E^{\bar{\mathcal{A}}}(s,\rho)$ till their total cost exceeds $\bar{c}(s,\rho)$.
4. Starting from the bottom end-point of s, keep on selecting edges of $E'(s,\rho)$ in the set $E^{\bar{\mathcal{A}}}(s,\rho)$ till their total cost exceeds $\bar{c}(s,\rho)$.
5. Output $E^{\bar{\mathcal{A}}}(s,\rho)$

Figure 3: Algorithm $\mathcal{B}$ for selecting edges in $E(s,\rho)$.

Lemma 5.4. *The total cost of edges selected by $\bar{\mathcal{A}}$ is at most 64 times that of $\mathcal{A}$. Further $\bar{\mathcal{A}}$ selects a feasible set of edges.*

Proof. For each cell (s,ρ), the algorithm in Figure 3 spends at most $2\bar{c}(s,\rho)$ in each of the steps 3 and 4 (the last edge selected in each of these steps could have cost up to $\bar{c}(s,\rho)$). The first statement now follows from Lemma 5.3. To prove feasibility, consider a cell (s,ρ) and an input path $P \in \mathcal{P}$ which contains at least one edge in $E^{\mathcal{A}}(s,\rho)$. Using assumption (P3), P either contains the top vertex or the bottom vertex of s – assume the former case. Let the edges of s from top to bottom be $e_1,\ldots,e_k$. The following fact is easy to prove by induction on i – the total size of edges from $\{e_1,\ldots,e_i\} \cap E(s,\rho)$ selected by $\bar{\mathcal{A}}$ is at least that of $\mathcal{A}$. For base case, if $\mathcal{A}$ picks $e_1 \in E(s,\rho)$, then $c_{e_1} \leq c^{\mathcal{A}}(s,\rho) \leq \bar{c}^{\mathcal{A}}(s,\rho)$ and so, $\mathcal{A}$ will also pick this edge (in Step 3 of Algorithm $\mathcal{B}$). Suppose the statement is true for $\{e_1,\ldots,e_i\}$. If $e_{i+1} \notin E(s,\rho)$ or $\mathcal{A}$ does not pick e_{i+1}, there is nothing to prove. So assume $\mathcal{A}$ picks this edge and $e_{i+1} \in E(s,\rho)$. Again, we are done if $\bar{\mathcal{A}}$ also picks this edge. If $\bar{\mathcal{A}}$ does not pick this edge, it has already exhausted its budget before it reaches this edge.

Since all edges in $E(s,\rho)$ have the same density and the budget of $\bar{\mathcal{A}}$ is more than that of $\mathcal{A}$ for any cell (s,ρ), it follows that the edges in $\{e_1,\ldots,e_i\} \cap E(s,\rho)$ selected by $\bar{\mathcal{A}}$ have more total size than the total size of edges in $E(s,\rho)$ selected by $\mathcal{A}$. This proves the induction hypothesis. Since it holds for all ρ, it follows that for all i, the total size of edges in $\{e_1,\ldots,e_i\}$ selected by $\bar{\mathcal{A}}$ is at least that by $\mathcal{A}$. Since P contains e_1, the edges of P in s are of the form $e_1,\ldots,e_i$ for some i. Therefore, total size of edges in $P \cap s$ selected by $\bar{\mathcal{A}}$ is at least that by $\mathcal{A}$. Summing over all segments, we see that $\bar{\mathcal{A}}$ also satisfies the demand of P. □

Modifying $\bar{\mathcal{A}}$ to $\widetilde{\mathcal{A}}$

We now modify $\bar{\mathcal{A}}$ to another algorithm $\widetilde{\mathcal{A}}$ which spends even more amount on low density edges. For a segment s, let $B^{\bar{\mathcal{A}}}(s)$ denote the maximum over all density classes ρ of $c^{\bar{\mathcal{A}}}(s,\rho)$. Note that $B^{\bar{\mathcal{A}}}(s)$ is also a power of 2. For each segment s, $\widetilde{\mathcal{A}}$ incurs an extra $4B^{\bar{\mathcal{A}}}(s)$ amount of cost on buying low density edges in s. The algorithm is described formally in Figure 4. It sorts the edges in s according to density and picks them in this order. We shall use $E^{\widetilde{\mathcal{A}}}(s,\rho)$ be the edges in s of density class ρ which are selected by $\widetilde{\mathcal{A}}$. We use $E^{\texttt{new}}(s)$ to denote the new edges in s which are selected by the algorithm $\widetilde{\mathcal{A}}$. Note that $\rho^\star(s)$ denotes the highest density class of an edge in $E^{\texttt{new}}(s)$ if we do not exhaust the budget in Step 3, otherwise it is set to $\Delta_{\max}+1$, one plus the highest density class. The algorithm ensures in Step 5 that the total cost of new edges of density class $\rho^\star(s)$ selected by the algorithm is at least $B^{\bar{\mathcal{A}}}(S)$ (or we select all such edges). This is done for the following reason – given $\rho^*(s)$, $\bar{c}(s,\rho^\star(s))$ and $B^{\bar{\mathcal{A}}}(s)$, we can figure out which edges in $E(s,\rho)$ are selected by $\widetilde{\mathcal{A}}$.

Algorithm $\widetilde{\mathcal{A}}$:

Input: A segment s, edges in s selected by $\bar{\mathcal{A}}$, and budget $B^{\bar{\mathcal{A}}}(s)$.

1. Let $E'(s)$ be the edges in s of cost at most $B^{\bar{\mathcal{A}}}(s)$ which are not selected by $\bar{\mathcal{A}}$
2. Let $e_1,\ldots,e_k$ be the edges in $E'(s)$ arranged according to increasing density [a]
3. Select edges in this order till their cost exceeds $B^{\bar{\mathcal{A}}}(s)$ (or we select all)
4. Define $\rho^\star(s)$ as the density class of the last edge e_i picked in this manner – it is set to $\Delta_{\max}+1$ if we select all edges in $E'(s)$ and the budget is not exhausted.
5. Continue to pick edges $e_{i+1},\ldots$ as long as their density class remains $\rho^\star(s)$ and the total cost of selected edges in $E'(s)$ of density class $\rho^\star(s)$ is at most $B^{\bar{\mathcal{A}}}(S)$.

[a]Edges of same density class are ordered in an arbitrary but fixed manner.

Figure 4: Algorithm $\widetilde{\mathcal{A}}$ for a segment s

Claim 5.5. *The total cost of edges selected by $\widetilde{\mathcal{A}}$ is at most a constant times that of $\mathcal{A}$.*

Proof. Since $\widetilde{\mathcal{A}}$ spends an additional amount of at most $4B^{\bar{\mathcal{A}}}(s)$ on each segment s, the result follows from Lemma 5.4. □

We now some interesting properties of $\widetilde{\mathcal{A}}$ which will yield the DP algorithm. We give some definitions. We say that a cell (s,ρ) *dominates* a cell (s',ρ') if s' is an ancestor of s, and if s' lies i levels above s, then $\rho' \geq \rho + i$. One way to visualize this definition is in the form of a table shown in Figure 5. For a segment s, let $s_0 = s, s_1, \ldots, s_k$ be the segments as we traverse from s to the root. For each s_i, we draw a column in the table with one entry for each cell (s_i,ρ), with ρ

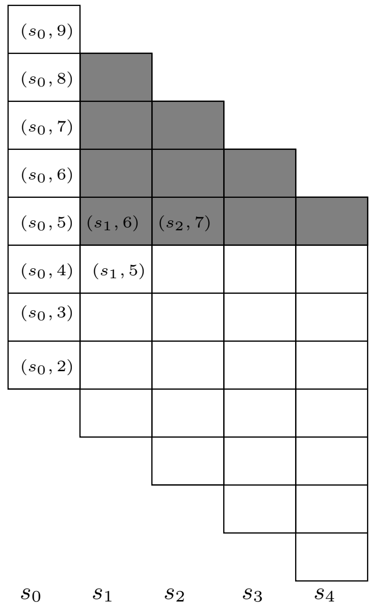

Figure 5: Cells (s_i, ρ) arranged in a table with $\Delta_{\min} = 2, \Delta_{\max} = 8$. The shaded region shows the cells dominated by $(s_0, 5)$.

increasing as we go up. Further as we go right, we shift these columns one step down. So row τ of this table will correspond to cells $(s_0, \tau), (s_1, \tau + 1), (s_2, \tau + 2)$ and so on. With this picture, a cell (s, ρ) dominates all cells which lie in the upper right quadrant with respect to the table entry containing (s, ρ). For a segment s, let $D(s)$ be the set of cells dominated by $(s, \rho^\star_s)$, and we use $E^{\widetilde{\mathcal{A}}}(D(s))$ to denote the edges selected by $\widetilde{\mathcal{A}}$ lying in these cells, i.e., $\cup_{(s',\rho') \in D(s)} E^{\bar{\mathcal{A}}}(s,', \rho')$. The following lemma shows why this notion is useful.

Lemma 5.6. *The total size of edges in $E^{\texttt{new}}(s)$ is at least the total size of edges in $E^{\bar{\mathcal{A}}}(D(s))$.*

Proof. Let $s_0 = s, s_1, \ldots, s_k$ be the segments as we traverse from s to the root. We begin with a fact which follows from the smoothness property.

Claim 5.7. *Then $B^{\bar{\mathcal{A}}}(s_i) \leq 8^i B^{\bar{\mathcal{A}}}(s)$, for $i = 1, \ldots, k$.*

Proof. It is enough to show this for s_1, the rest will follow by induction. Suppose $B^{\bar{\mathcal{A}}}(s_1) = c^{\bar{\mathcal{A}}}(s_1, \rho)$ for some density class ρ. Then $B^{\bar{\mathcal{A}}}(s) \geq c^{\bar{\mathcal{A}}}(s, \rho) \geq c^{\bar{\mathcal{A}}}(s_1, \rho)/8 = B^{\bar{\mathcal{A}}}(s_1)/8$, where we used the smoothness property in Lemma 5.3. □

For sake of brevity, let ρ denote $\rho^\star(s)$, and B denote $B^{\bar{\mathcal{A}}}(s)$. We assume that $\rho \neq \Delta_{\max} + 1$, otherwise $D(s)$ is empty. Since the total cost of edges in $E^{\texttt{new}}(s)$ is at least B and each of these has density at most 16^ρ, the total size of these edges is at least $16^{-\rho} \cdot B$. Consider the segment s_i. Then $D(s)$ contains the cells (s_i, ρ'), where $\rho' \geq \rho + i$. The total size of edges selected by $\bar{\mathcal{A}}$ in any such cell is at most $B^{\bar{\mathcal{A}}}(s_i) \cdot 16^{-\rho'} \leq 8^i \cdot 16^{-\rho'} \cdot B$ (using Claim 5.7). Therefore the total size of the edges in $\cup_{\rho':\rho' \geq \rho+i} E^{\bar{\mathcal{A}}}(s_i, \rho')$ is at most

$$\sum_{\rho' \geq \rho+i} 8^i \cdot 16^{-\rho'} \cdot B \leq 2 \cdot 16^{-\rho} \cdot B/8^i.$$

Summing over all $i \geq 1$ yields the lemma. $\square$

We say that an edge $e \in E^{\widetilde{\mathcal{A}}}$ *covers* a demand path P if the total size of edges in $P \cap E^{\widetilde{\mathcal{A}}}$ which lie below e is less than the demand of P. In other words, P is covered by the lowermost edges selected by $\widetilde{\mathcal{A}}$. The following corollary follows directly from Lemma 5.6 and the feasibility of $\bar{\mathcal{A}}$.

Corollary 5.8. *Let s be a segment with lower end-point being $v \in V(\mathcal{T}')$. Let P be a path containing v. Then P is not covered by any edge in $E^{\widetilde{\mathcal{A}}}(D(s))$.*

The statement above says that as far as v is concerned it can ignore all the edges in $E^{\widetilde{\mathcal{A}}}(D(s'))$, where s' is an ancestor of s (including s). We now show that such edges can be described in a simple manner. Consider a vertex $v \in V(\mathcal{T}')$ and let $s_1, s_2, \ldots, s_k$ be the segments encountered as we go from v to the root. For sake of brevity, let $\rho_i^\star$ denote $\rho_{s_i}^\star$. We define a sequence of cells $Q^{\widetilde{\mathcal{A}}}(v)$ as follows: it starts with the cell $(s_1, \Delta_{\max}+1)$, and goes down till $(s_1, \rho_1^\star)$. After this it keeps moving right (with respect to the interpretation as in Figure 5) till it reaches a segment s_i where $\rho_i^\star$ is below the current cell occupied by the sequence and then moves down and so on. This is described formally in Figure 6 (note that the sequence may contain a cell of the form $(s_i, \Delta_{\max}+1)$, and the last cell is $(s_k, \rho_k^\star)$). An example is given in Figure 7.

Construct Sequence $Q^{\widetilde{\mathcal{A}}}(v)$:

Input: A node $v \in \mathcal{T}'$ at depth k, integers $\rho_1^\star, \ldots, \rho_k^\star$

1. Initialise $Q^{\widetilde{\mathcal{A}}}(v)$ to empty sequence, and $i \leftarrow 1, \rho \leftarrow \Delta_{\max}+1$
2. While $(i \leq k)$
   (i) Add the cell (s_i, ρ) to $Q^{\widetilde{\mathcal{A}}}(v)$.
   (ii) If $\rho > \rho_i^\star$, $\rho \leftarrow \rho - 1$
   (iii) Else $i = i+1, \rho \leftarrow \rho + 1$.

Figure 6: Construction of the path $Q^{\widetilde{\mathcal{A}}}(v)$.

For a segment s_i, the density classes ρ for which (s_i, ρ) belong to $Q^{\widetilde{\mathcal{A}}}(v)$ form an interval $[\rho_l, \rho_r]$. We say that cell (s_i, ρ) is *above* (or *below*) $Q^{\widetilde{\mathcal{A}}}(v)$ if $\rho > \rho_r$ (or $\rho < \rho_l$). The sequence $Q^{\widetilde{\mathcal{A}}}(v)$ has the following property.

Claim 5.9. *If a cell (s_i, ρ) lies above $Q^{\widetilde{\mathcal{A}}}(v)$ then it belongs to the set $D(s_j)$ for some segment s_j (which is an ancestor of v). If a cell (s_i, ρ) lies below $Q^{\widetilde{\mathcal{A}}}(v)$, then $\rho < \rho_i^\star$.*

Proof. Suppose a cell (s_i, ρ) lies above this sequence. Let j be the first index before i such that $Q^{\widetilde{\mathcal{A}}}(v)$ contains the cell $(s_j, \rho_j^\star)$ ($j = 1$ is always an option). It follows that we never reach Step 2(ii) in Algorithm 6 after adding this cell to $Q^{\widetilde{\mathcal{A}}}(v)$ and before reaching segment s_i. Therefore $Q^{\widetilde{\mathcal{A}}}(v)$ contains the cell $(s_i, \rho_j^\star + (i-j))$. But then $\rho > \rho_j^\star + (i-j)$, and therefore, (s_i, ρ) is dominated by $(s_j, \rho_j^\star)$. This proves the first part. For the second part, suppose (s_i, ρ) is below $Q^{\widetilde{\mathcal{A}}}(v)$ and let (s_i, ρ') be the first cell in $Q^{\widetilde{\mathcal{A}}}(v)$ which corresponds to segment s_i. Two cases can occur: (i) $\rho' \leq \rho_i^\star$ in which case $\rho < \rho_i^\star$ as well, (ii) $\rho' > \rho_i^\star$, in which case we add $(s_i, \rho_i^\star)$ to the sequence as well, and so, $\rho < \rho_i^\star$. $\square$

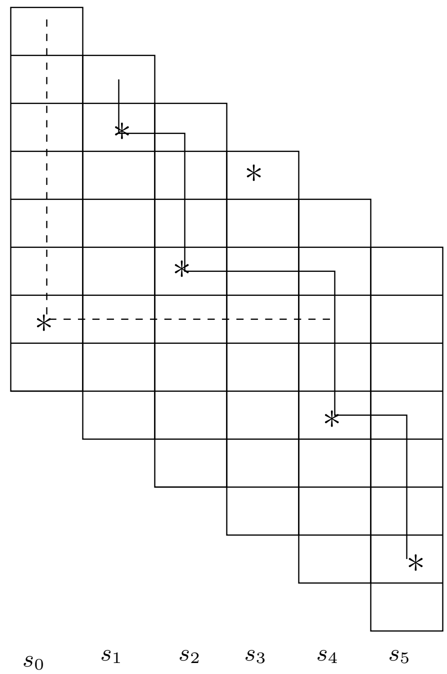

Figure 7: Vertex v has segments $s_1, \ldots, s_5$ as ancestors, and $Q^{\widetilde{\mathcal{A}}}(v)$ is shown in solid lines (stars denote the location of corresponding $\rho_i^\star$ values). The vertex v has a child w in $\mathcal{T}'$, and the dotted lines show $Q^{\widetilde{\mathcal{A}}}(w)$ till it merges with $Q^{\widetilde{\mathcal{A}}}(v)$.

The claim above shows that given $Q^{\widetilde{\mathcal{A}}}(v)$, we can easily figure out how paths through v are getting covered. If a cell (s_i, ρ) lies above this sequence, we can ignore the edges $E(s_i, \rho)$. If it lies below $Q^{\widetilde{\mathcal{A}}}(v)$, we know that $\widetilde{\mathcal{A}}$ selects all the edges in $E(s_i, \rho)$ of cost at most $B^{\bar{\mathcal{A}}}(s_i)$ (because $\widetilde{\mathcal{A}}$ did not stop at this density class while running the algorithm in Figure 4). How about cells (s_i, ρ) which are on $Q^{\widetilde{\mathcal{A}}}(v)$? If $\rho > \rho_i^\star$, then $E^{\widetilde{\mathcal{A}}}(s_i, \rho)$ is same as $E^{\bar{\mathcal{A}}}(s_i, \rho)$. If $\rho = \rho_i^\star$, then $E^{\widetilde{\mathcal{A}}}(s_i, \rho) \setminus E^{\bar{\mathcal{A}}}(s_i, \rho)$ is obtained by selecting edges from $E(s_i, \rho)$ as described in Figure 4 – consider edges of cost at most $B^{\bar{\mathcal{A}}}(s_i)$ in $E(s_i, \rho) \setminus E^{\bar{\mathcal{A}}}(s_i, \rho)$, and select them in order of increasing density till the total cost of selected edges exceeds $B^{\bar{\mathcal{A}}}(s_i)$. We are now ready to state the algorithm.

5.3 The Dynamic Program

In this section, we describe the dynamic program. We assume that the cost of the edges are integers and lie in the range $[1, n^2]$ (see property (P1)). We begin with some definitions. A sequence of integers $a_1, a_2, \ldots$ is said to be *smooth* if each a_i is a power of 2, and for all $i \geq 2$, the ratio a_i/a_{i-1} lies in the range $[1/64, 64]$. For example, the sequence $B^{\bar{\mathcal{A}}}(\rho), \rho \in [\Delta_{\min}, \Delta_{\max}]$, is smooth (by an argument similar to Claim 5.7).

Let v be a vertex in $V(\mathcal{T}')$, and $s_1, \ldots, s_k$ be the segments encountered as we go from v to the root. A sequence of cells $Q(v)$ is said to be *canonical* if it starts with $(s_1, \Delta_{\max}+1)$, and follows the following rules: (i) If (s_i, ρ) appears in $Q(v)$, then the next cell is either $(s_i, \rho - 1)$, or $(s_{i+1}, \rho + 1)$, (ii) the last cell in the sequence is of the form (s_k, ρ) for some density class ρ, and (iii) ρ lies in the range $[\Delta_{\min}, \Delta_{\max} + 1]$. As an example, $Q^{\widetilde{\mathcal{A}}}(v)$ constructed in the previous section is canonical.

Let v be a vertex in $V(\mathcal{T}')$ with $s_1, \ldots, s_k$ be the segments on the path from v to the root in $\mathcal{T}$. In the DP table D, we will have entries $D[v, B_1, \ldots, B_k, Q(v), C(v), T(v)]$, where $Q(v) = \sigma_1, \ldots, \sigma_\ell$ is a canonical sequence of cells, $B_1, \ldots. B_k$ and $C_{\sigma_1}, \ldots, C_{\sigma_\ell}$ are smooth sequences, with B_1, C_{σ_1} being in the range $[1, n^2]$, and $T(v)$ is a boolean array indexed by $Q(v)$.

To understand what this entry stores, we first describe an algorithm to select edges from $s_1, \ldots, s_k$ – these will only be a subset of edges from $s_1, \ldots. s_k$ which will be selected by our algorithm, but they will be only ones relevant for paths which go through v. This algorithm mimics what $\widetilde{\mathcal{A}}$ would do with the corresponding parameters $B_1^{\bar{\mathcal{A}}}, \ldots, B_k^{\bar{\mathcal{A}}}, Q^{\widetilde{\mathcal{A}}}(v)$, $C(v)$ being the costs $\bar{c}$ for the cells in $Q^{\widetilde{\mathcal{A}}}(v)$, and $T(v)$ being the boolean array which tells whether a cell (s_i, ρ) in $Q^{\widetilde{\mathcal{A}}}(v)$ satisfies $\rho = \rho_i^\star$ or not. The edge selection algorithm is shown in Figure 8. Note that we can define whether a cell (s_i, ρ) lies above or below $Q(v)$ in exactly the same manner as for $Q^{\widetilde{\mathcal{A}}}(v)$.

Select Edges :

Input: A node $v \in \mathcal{T}'$ at depth $k, B_1, \ldots, B_k, Q(v), C(v), T(v)$ as in the DP table.

1. Initialize a set X to $\emptyset$.
2. For $i = 1, \ldots, k$

   For $\rho = \Delta_{\min}, \ldots, \Delta_{\max}$

   (i) If the cell (s_i, ρ) lies above $Q(v)$, do nothing.

   (ii) Else If the cell (s_i, ρ) lies below $Q(v)$

   Add to X all edges in $E(s_i, \rho)$ of cost at most B_i.

   (iii) Else (i.e., the cell (s_i, ρ) belongs to $Q(v)$)

   (a) Add to X edges from $E(s_i, \rho)$ by running Algorithm $\mathcal{B}$ in Figure 3 with budget $C(v)_{(s_i, \rho)}$ – call these edges $E'(s_i, \rho)$.

   (b) If $T(v)_{(s_i, \rho)}$ is true

   -Let $e_1, \ldots, e_r$ be the edges of cost at most B_i in $E(s_i, \rho) \setminus E'(s_i, \rho)$ [a]

   -Keep adding to X these edges till their total cost exceeds B_i.
3. Return X.

[a]arranged in the same order as used in Algorithm $\widetilde{\mathcal{A}}$ in Figure 4

Figure 8: Edge Selection Algorithm.

We can now explain what a table entry $D[v, B_1, \ldots, B_k, Q(v), C(v), T(v)]$ stores. Let $\mathcal{P}(v)$ be the paths which include v as an internal vertex. It stores the cost of a solution for the instance defined by the subtree rooted at v for the paths in $\mathcal{P}(v)$ provided we also select edges from $s_1, \ldots, s_k$ as given by the algorithm in Figure 8 (the cost of these edges is not added in the table entry). If these parameters happen to be the ones used by $\widetilde{\mathcal{A}}$, then the this entry has value at most the cost of edges selected by $\widetilde{\mathcal{A}}$ in the subtree below v.

We need to specify one more sub-routine before we can describe the DP algorithm. Let w be a child of v in $\mathcal{T}'$, and let s_0 denote the segment joining w to v. We would like to understand how to update $Q(v)$ to $Q(w)$. Note that $Q(v)$ is a sequence which is supposed to be given by values $\rho_1^\star, \ldots, \rho_k^\star$ (even though we do not explicitly mention these in the table index parameters). Then to define $Q(w)$, we just need one more parameter $\rho_{s_0}^\star$, which we will denote by $\rho_0^\star$. This routine is given in Figure 9. The algorithm here takes as input the sequence $Q(v)$, an integer ρ_0 between $\Delta_{\min}$ and $\Delta_{\max} + 1$, and the boolean sequence $T(v)$ (also see Figure 7 for a pictorial representation of the algorithm). It starts building the sequence $Q(w)$ from the cell $(s_0, \Delta_{\max}+1)$, and decreases the density class of the cell till it reaches (s_0, ρ_0). After this it moves right till it hits $Q(v)$. It is easy to see that this is exactly how $Q^{\widetilde{\mathcal{A}}}(v)$ will get updated to $Q^{\widetilde{\mathcal{A}}}(w)$. The boolean sequence $T(w)$ is false for all these new cells except for (s_0, ρ_0).

Extend Sequence :

Input: Node $v \in \mathcal{T}', Q(v), T(v)$ as in the DP table, a child w of v in $\mathcal{T}'$,
$\rho_0 \in [\Delta_{\min}, \Delta_{\max}+1]$,

1. Let $\sigma := (s_0, \Delta_{\max}+1)$ be the current cell. Set i to 0.
2. Initialize $Q(w)$ to emptyset.
3. While $i \leq k$ and $\sigma = (s_i, \rho)$ is not in $Q(v)$
   (a) Add σ to $Q(w)$.
   (b) If $i = 0$ and $\rho > \rho_0$
   set $T(w)_\sigma$ to false, and update σ to $(s_0, \rho - 1)$.
   (c) Else if $i = 0$ and $\rho = \rho_0$
   set $T(w)_\sigma$ to true, and update σ to $(s_1, \rho+1)$, i to 1.
   (d) Else set $T(w)_\sigma$ to false, and update σ to $(s_{i+1}, \rho+1)$, i to $i+1$.
4. Append to $Q(w)$ and $T(w)$ the subsequence of $Q(v)$ and $T(v)$ after σ
5. Return $Q(w)$ and $T(w)$.

Figure 9: Extending $Q(v)$ to $Q(w)$. Note that s_0 denotes the segment joining v and w.

The complete details of how to fill a table entry is shown in Figure 10. These entries are filled in a bottom up manner. We use M_1 and M_2 to keep the minimum value of solution corresponding to the sub-instances for the two children. In Step 4(i), we guess the ρ_0^r and B_0^r values for the segment joining v to the child w_r. In Step 4(ii), we guess the cost spent on each density class in the corresponding segment. In Step 4(ii) (a), we selects edges in the segment s_0^r corresponding to this cost vector. We do exactly what $\widetilde{\mathcal{A}}$ would do – note that $\bar{\mathcal{A}}$ just uses c_ρ^r to figure out which edges to select (in Figure 3) and then $\widetilde{\mathcal{A}}$ extends this using the knowledge of ρ_0^r, B_0^r only (in Figure 4). In step 4(ii)(b), we check whether all paths ending in s_0^r are satisfied. Then we call Algorithm **Extend Sequence** to figure out $Q(w_r)$ and the associated boolean array $T(w_r)$. Finally we need to figure out the costs assigned to each cell in $Q(w_r)$. For cells corresponding to s_0^r, we have already done this in Step 4(ii). For the next part of $Q(w_r)$ till it meets $Q(v)$ we guess these values in Step 4(ii)(g). Finally, we look at the corresponding entries in the tables for the two children.

We now analyze this algorithm.

5.3.1 Running Time Analysis

The running time analysis follows easily from counting the number of smooth sequences. Note that the number of possible smooth sequences $a_1, \ldots, a_k$ of length k would be at most $2^{O(k)}$. This is so because a_1 has $O(\log n)$ choices (as it is a power of 2, and lies in the range $[1, n^2]$). Given a_i, smoothness property shows that a_{i+1} has to lie in the range $[a_i/64, 64a_i]$. Since a_{i+1} is also a power of 2, there are only constant number of choices for a_{i+1}. Notice that the sequences $B_1, \ldots, B_k$, and $C(v)$ are required to be smooth. Since k is $O(\log(nP))$ (Lemma 5.2), it follows that the number of possible choices for $B_1, \ldots, B_k$ is $(nP)^{O(1)}$, and similarly for $C(v)$.

Fill DP Table :

Input: A node $v \in \mathcal{T}'$ at depth $k, B_1, \ldots, B_k, Q(v), C(v), T(v)$ as explained in the text.

0. If v is a leaf node return 0.
1. Let X be the edges selected by algorithm **Select Edges** in Figure 8 when given parameters $v, B_1, \ldots, B_k, Q(v), C(v), T(v)$.
2. Let w_1, w_2 be the two children of v in $\mathcal{T}'$ and the corresponding segments be s_0^1, s_0^2.
3. Initialize M_1, M_2 to ∞.
4. For $r = 1, 2$ (go to each of the two children and solve the subproblems)
   - (i) For each $\rho_0^r \in \{\Delta_{\min}, \ldots, \Delta_{\max} + 1\}, B_0^r \in [1, n^2]$ [a] do
   - (ii) For each smooth sequence $(c^r_{\Delta_{\min}}, c^r_{\Delta_{\min}+1}, \ldots, c^r_{\Delta_{\max}})$, with $c^r_\rho \leq B_0^r$ do
     - (a) Let X^r be the edges in s_0^r which would get selected if we run $\widetilde{\mathcal{A}}$ (along with $\bar{\mathcal{A}}$) with parameters c^r_ρ, B_0^r
     - (b) If any path in $\mathcal{P}$ ending in the segment s_0^r is not satisfied by the edges in $X \cup X^r$ exit this loop
     - (c) Let $Q(w_r), T(w_r)$ be the sequences returned by **Extend Sequence** in Figure 9 when given parameters $v, Q(v), T(v), w_r, \rho_0^r$.
     - (d) Let $\sigma_1, \ldots, \sigma_\ell$ be the cells in $Q(w)$ starting after (s_0^r, ρ_0^r) and ending before we reach a cell in $Q(v)$ (so, $\sigma_i = (s_i, \rho_0^r + i)$).
     - (e) Let $\sigma_{\ell+1}$ be the next cell in $Q(w_r)$ and so, $\sigma_{\ell+1} \in Q(v)$.
     - (f) Let $C'(v)$ be the subsequence of $C(v)$ after (and including) $\sigma_{\ell+1}$
     - (g) For all $c_{\sigma_1}, \ldots, c_{\sigma_\ell}$, where $c_{\sigma_i} \leq B_i$, such that $C(w_r) := (c^r_{\Delta_{\max}}, \ldots, c^r_{\rho_0^r}, c_{\sigma_1}, \ldots, c_{\sigma_\ell}, C'(v))$ is smooth [b], $M_r \leftarrow \min(M_r, \text{cost of } X^r + D[w_r, B_0^r, B_1 \ldots, B_k, Q(w_r), C(w_r), T(w_r)])$.
5. $D[v, B_1, \ldots, B_k, Q(v), C(v), T(v)] \leftarrow M_1 + M_2$.

[a] B_0^r should be a power of 2, and $B_0^r, B_1, \ldots, B_k$ should be a smooth sequence

[b] First entry of $C(w_r)$ should be 0 because the first cell in $Q(w_r)$ is $(s_0^r, \Delta_{\max} + 1)$, this is assumed implicitly

Figure 10: Filling a table entry $D[v, B_1, \ldots, B_k, Q(v), C(v), T(v)]$ in the dynamic program.

Claim 5.10. *The length of any canonical sequence $Q(v)$ for a vertex $v \in \mathcal{T}'$ is $O(\log(nP))$, and so, there are pseudo-polynomial many choices for $Q(v), T(v)$.*

Proof. Let $s_1, \ldots, s_k$ be segments defined as in the previous section. For a cell (s_i, ρ) define its potential $\phi(s_i, \rho)$ as $2i - \rho$. For the first cell in $Q(v)$, its potential is $1 - \Delta_{\max}$, and for the final cell, it is at most $2k - \Delta_{\min}$. When we go from one cell to the next in $Q(v)$, ϕ strictly increases by at least 1 unit. Therefore its length is at most $2k + \Delta_{\max} - \Delta_{\min}$. Since both k and $(\Delta_{\max} - \Delta_{\min})$ are $O(\log(nP))$, the first result follows. Since $Q(v)$ is a sequence of length $O(\log(nP))$, where at each step there can be only two choices (decrease ρ or increase i), it follows that there only pseudo-polynomially many choices for $Q(v)$. □

The claim above shows that there are pseudo-polynomial number of table entries. Now let us see the time taken to fill a table entry (as in Figure 10). Again notice that we just cycle over constant number of smooth sequences, each of $O(\log(nP))$ length. Therefore, we conclude

Theorem 5.11. *The Dynamic Programming Algorithm runs in pseudo-polynomial time.*

5.3.2 Correctness and Approximation Ratio

We first argue that the table entries in the DP correspond to valid solutions. Fix a vertex $v \in V(\mathcal{T}')$, and let $s_1, \ldots, s_k$ be the segments as we go from v to the root. Let $\mathcal{P}(v)$ be the paths in the input which contain v as an internal vertex.

Lemma 5.12. *For valid parameters $B_1, \ldots, B_k, Q(v), C(v), T(v)$ in the DP table, let X be the set of edges in segments $s_1, \ldots, s_k$ selected by the Algorithm* **Select Edges** *in Figure 8. Then the table entry $D[v, B_1, \ldots, B_k, Q(v), C(v), T(v)]$ denotes the cost of a subset Y of edges lying the sub-tree of $\mathcal{T}$ rooted at v such that $Y \cup X$ is a feasible solution for the demands in $\mathcal{P}(v)$.*

Proof. We prove this by induction on the height of v. If v is a leaf, then $\mathcal{P}(v)$ is empty, and so the result follows trivially. Suppose it is true for all nodes in $\mathcal{T}'$ at height h, and v be at height $h+1$ in $\mathcal{T}'$. We use the notation in Figure 10. Let w_1, w_2 be the two children of v. Let r denote either 1 or 2. Let the value of M_r used in Step 5 be equal to $D[w_r, B_0^r, \ldots, B_k, Q(w_r), C(w_r), T(w_r)]$. Also let $\rho_0^r, c^r_{\Delta_{\min}}, \ldots . c^r_{\Delta_{\max}}$ be the parameters found in Steps 4(i), (ii) respectively. Let X and X^r be as in the steps 1 and 4(ii)(a) respectively. We ensure that $X \cup X^r$ covers all paths in $\mathcal{P}(v)$ which end before w^r. The following claim is the key to the correctness of the algorithm.

Claim 5.13. *Let $X(w_r)$ be the edges selected by the Algorithm* **Select Edges** *in Figure 8 when called with parameters $w_r, B_0^r, B_1, \ldots, B_k, Q(w_r), C(w_r), T(w_r)$. Then $X(w_r)$ is a subset of $X \cup X^r$.*

Proof. First consider the segment s_0^r. The set X^r contains all edges in s_0^r that get selected if we run $\widetilde{\mathcal{A}}$ (along with $\bar{\mathcal{A}}$) on this segment with parameters $c^r_{\Delta_{\min}}, \ldots, c^r_{\Delta_{\max}}, B_0^r$, whereas $X(w_r) \cap s_0^r$ will contain a subset of these edges corresponding to density classes $\Delta_{\min}, \ldots . \rho_0^r$. So, $X(w_r) \cap s_0^r \subseteq X^r$. Let us now worry about segments $s_i, i \geq 1$. Fix such a segment s_i.

We know that after segment s_0, the sequence $Q(w)$ lies below $Q(v)$ till it meets $Q(v)$ (because $Q(v)$ starts with $(s_1, \Delta_{\max}+1)$). Now consider various cases in the Algorithm **Select Edges**. If a cell (s_i, ρ) lies above $Q(w)$, we do not pick any edges from $E(s_i, \rho)$ in $X(w_r)$. So we need not worry about such cells. If it lies below $Q(w)$, it will also lie below $Q(v)$, and so X and $X(w_r)$ will select the same edges from $E(s_i, \rho)$. Finally, consider the case when it lies on $Q(w)$. Again, if it lies on $Q(v)$, both X and $X(w_r)$ will agree on this cell. So suppose it does not lie on $Q(v)$, in which case, it will lie below $Q(v)$. In the set X, we will pick all the edges in $E(s_i, \rho)$ of cost at most B_i, which will clearly be a superset of what $X(w_r)$ picks (because this cell will be one of the cells of the form σ_i as in Step 4(ii)(d), c_{σ_i} guessed in Step 4(ii)(g) is at most B_i, and $T(w_r)_{\sigma_i}$ is false). This proves the claim. ☐

By induction hypothesis, there is a subset of $Y(w_r)$ of edges in the subtree rooted at w_r of cost $D[w_r, B_0^r, B_1 \ldots, B_k, Q(w_r), C(w_r), T(w_r)]$ such that $Y(w_r) \cup X(w_r)$ covers all paths in $\mathcal{P}(w_r)$. We already know that $X \cup X^r$ covers all paths in $\mathcal{P}(v)$ which end in segment s_0^r. Since any path in $\mathcal{P}(v)$ will either end in s_0^1 or s_0^2, or will belong to $\mathcal{P}(w_1) \cup \mathcal{P}(w_2)$, it follows that all paths in $\mathcal{P}(v)$ are covered by $\cup_{r=1}^2 (Y(w_r) \cup X(w_r) \cup X^r) \cup X$. Now, the claim above shows that $X(w_r) \subseteq X^r \cup X$. So this set is same as $Y(w_1) \cup Y(w_2) \cup X^1 \cup X^2 \cup X$ (and these sets are mutually disjoint). Since the DP table entry for v for these parameters is exactly the cost of $Y(w_1) \cup Y(w_2) \cup X^1 \cup X^2$, the result follows. ☐

We now relate the DP to the optimal solution $\mathcal{A}$. Again consider a vertex $v \in V(\mathcal{T}')$, and let $s_1, \ldots, s_k$ be the segments as we traverse from v to the root. We shall use $B_i^{\bar{\mathcal{A}}}$ to denote $B^{\bar{\mathcal{A}}}(s_i)$ and $\rho_i^\star$ to denote $\rho^\star(s_i)$. Let $Q^{\widetilde{\mathcal{A}}}(v)$ be the canonical sequence corresponding to $\widetilde{\mathcal{A}}$ at v. Let $\bar{C}(v)$ be the vector such that for a cell $\sigma \in Q^{\widetilde{\mathcal{A}}}(v)$, $\bar{C}(v)_\sigma$ stores $\bar{c}(\sigma)$ (note that it is $\bar{c}$ and not $c^{\widetilde{\mathcal{A}}}$ because we can argue about the smoothness of $\bar{c}$ only). Let $T^{\widetilde{\mathcal{A}}}(v)$ be the corresponding boolean sequence which is 1 whenever the corresponding cell is of the form $(s_i, \rho^\star_{s_i})$. Using arguments similar to Claim 5.7 and Lemma 5.3, it is easy to check that the sequences $B_1^{\bar{\mathcal{A}}}, \ldots, B_k^{\bar{\mathcal{A}}}$, and $\bar{C}(v)$ are smooth.

Lemma 5.14. *The table entry $D[v, B_1^{\bar{\mathcal{A}}}, \ldots, B_k^{\bar{\mathcal{A}}}, Q^{\widetilde{\mathcal{A}}}(v), \bar{C}(v), T^{\widetilde{\mathcal{A}}}(v)]$ is at most the cost of edges selected by $\widetilde{\mathcal{A}}$ in the subtree rooted at v.*

Proof. We prove by induction on height of v. It is trivial when v is a leaf. So, let v be a node in $V(\mathcal{T}')$ and assume the statement is true for the children w_1, w_2 of v (in $\mathcal{T}'$). Consider the iteration when we try $\rho_0 = \rho_0^\star, c_\rho^r = \bar{c}(s_0^r, \rho)$ for all density classes ρ (note that the sequence $\bar{c}(s_0^r, \rho)$ is a smooth sequence by Lemma 5.3). In this iteration $Q(w_r)$ will be same as $Q^{\widetilde{\mathcal{A}}}(w_r)$. Consider the choices for $c_{\sigma_1}, \ldots, c_{\sigma_\ell}$ to be the budgets $\bar{c}$ on these cells (in Step 4(ii)(g)).

In such settings of parameters X^r will be exactly the edges in s_0^r selected by $\widetilde{\mathcal{A}}$. Further, by induction hypothesis, $D[w_r, B_0^{\bar{\mathcal{A}}}, \ldots, B_k^{\bar{\mathcal{A}}}, Q^{\widetilde{\mathcal{A}}}(w_r), \bar{C}(w_r), T^{\widetilde{\mathcal{A}}}(w_r)]$ will be at most the cost of edges selected by $\widetilde{\mathcal{A}}$ in the subtree below w_r. Since the DP considers minimum cost of several choices, the entry for this table entry for v will be at most the cost of edges selected by $\widetilde{\mathcal{A}}$ in the subtree below v. ☐

We are now done because if we consider v as the root (note that the parameters $B_1, \ldots, B_k, Q(v)$, etc. will be empty now), Lemma 5.14 states that $D[v]$ is at most the cost of $\widetilde{\mathcal{A}}$. But now, Lemma 5.12 states that $D[v]$ stores the cost of a feasible solution. Thus, we have a solution whose cost is at most that of $\widetilde{\mathcal{A}}$, and now we are done by Claim 5.5.

6 Discussion

We give the first pseudo-polynomial time constant factor approximation algorithm for the weighted flow-time problem on a single machine. The algorithm can be made to run in time polynomial in n and W as well, where W is the ratio of the maximum to the minimum weight. The rough idea is as follows. We have already assumed that the costs of the job segments are polynomially bounded (this is without loss of generality). Since the cost of a job segment is its weight times its length, it follows that the lengths of the job segments are also polynomially bounded, say in the range $[1, n^c]$. Now we ignore all jobs of size less than $1/n^2$, and solve the remaining problem using our algorithm (where P will be polynomially bounded). Now, we introduce these left out jobs, and show that increase in weighted flow-time will be small. We leave the problem of obtaining a truly polynomial time constant factor approximation algorithm as open.

References

[1] Nikhil Bansal and Kedar Dhamdhere. Minimizing weighted flow time. *ACM Trans. Algorithms*, 3(4):39, 2007.

[2] Nikhil Bansal and Kirk Pruhs. The geometry of scheduling. *SIAM J. Comput.*, 43(5):1684–1698, 2014.

[3] Deeparnab Chakrabarty, Elyot Grant, and Jochen Könemann. On column-restricted and priority covering integer programs. In *Integer Programming and Combinatorial Optimization, 14th*

International Conference, IPCO 2010, Lausanne, Switzerland, June 9-11, 2010. Proceedings, pages 355–368, 2010.

[4] Timothy M. Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1576–1585, 2012.

[5] Chandra Chekuri and Sanjeev Khanna. Approximation schemes for preemptive weighted flow time. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 297–305, 2002.

[6] Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 84–93, 2001.

[7] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.