# Minimizing the Maximum Flow Time in the Online Food Delivery Problem

**Xiangyu Guo**
University at Buffalo, NY, USA

**Kelin Luo**
Institute of Computer Science, Universität Bonn, Germany

**Shi Li**
University at Buffalo, NY, USA

**Yuhao Zhang**
Shanghai Jiao Tong University, China

## Abstract

We study a common delivery problem encountered in nowadays online food-ordering platforms: Customers order dishes online, and the restaurant delivers the food after receiving the order. Specifically, we study a problem where $k$ vehicles of capacity $c$ are serving a set of requests ordering food from one restaurant. After a request arrives, it can be served by a vehicle moving from the restaurant to its delivery location. We are interested in serving all requests while minimizing the maximum flow-time, i.e., the maximum time length a customer waits to receive his/her food after submitting the order.

We show that the problem is hard in both offline and online settings even when $k = 1$ and $c = \infty$: There is a hardness of approximation of $\Omega(n)$ for the offline problem, and a lower bound of $\Omega(n)$ on the competitive ratio of any online algorithm, where $n$ is number of points in the metric.

We circumvent the strong negative results in two directions. Our main result is an $O(1)$-competitive online algorithm for the uncapacitated (i.e, $c = \infty$) food delivery problem on tree metrics; we also have negative result showing that the condition $c = \infty$ is needed. Then we explore the speed-augmentation model where our online algorithm is allowed to use vehicles with faster speed. We show that a moderate speeding factor leads to a constant competitive ratio, and we prove a tight trade-off between the speeding factor and the competitive ratio.

**2012 ACM Subject Classification** Theory of computation → Online algorithms

**Keywords and phrases** Online algorithm, Capacitated Vehicle Routing, Flow Time Optimization

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2022.33

**Related Version** *Full Version*: `https://arxiv.org/abs/2110.15772`

**Funding** *Kelin Luo*: Supported by the National Natural Science Foundation of China, Grant No. 72071157.
*Yuhao Zhang*: Supported by the National Natural Science Foundation of China, Grant No. 62102251.

## 1 Introduction

Online food-ordering-and-delivery services (e.g., UberEats and Doordash) have become more and more popular in the past decade. Customers can order dishes online and wait for the restaurant to deliver the food to their home. Arguably one of the most important factors for service quality is the *flow time*, i.e., how long a customer waits to receive the food after submitting his/her order. We formalize the problem as the following *Online Food Delivery Problem* (Online FDP). Let $(V, d)$ be a metric space with $|V| = n$, where $d$ is a metric on $V$. Let $o \in V$ be the *depot* that has $k$ unit-speed vehicles, each with a capacity $c \in \mathbb{Z}_{>0} \cup \{\infty\}$. The requests arrive online, where a request $\rho = (r_\rho, v_\rho)$ is released at time $r_\rho$ with delivery location $v_\rho$. To serve $\rho$, a vehicle needs to first pick up the ordered food at the depot (after $\rho$ is released) and deliver it to $v_\rho$. The goal of the online FDP is to schedule vehicles to deliver the food to their delivery locations (i.e., to serve the requests), satisfying the requirement that a vehicle can only carry food for at most $c$ requests at any time. The objective of the problem is to minimize the maximum flow time. This model captures the real-world scenario that one restaurant owns several vehicles and needs to deliver food to customers, and the restaurant wants to make all customers satisfied, i.e, wait for a short period of time.

The *offline* food delivery problem (FDP) is closely related to many vehicle routing problems. For example, the uncapacitated (i.e., $c = \infty$) single-vehicle (i.e., $k = 1$) FDP with all requests released at time 0 already captures a variant of the traveling salesman path problem. When the vehicle has finite capacity $c$, the FDP just mentioned is also known as the Capacitated Vehicle Routing Problem (CVRP). A more general setting is studied under the name Dial-a-Ride Problem (DaRP), where besides the delivery location, each customer can specify its own pickup location. To satisfy the request, a vehicle has to take the customer from the pick-up location to its delivery location. Most results on offline CVRP/DaRP focused on the total travel distance objective [1, 24, 12, 23]; while in the online model, much effort has been devoted to completion time objectives like average completion time [25, 9] or makespan [19, 2]. Compared with previous studies, we are focusing on a much harder and less-studied objective – maximum flow time, in the online setting. The only theoretical results we know regarding maximum flow time are two lower bounds for online TSP and online DaRP: Krumke et al.[33] show that the single-vehicle *finite-capacity* DaRP does not admit $O(1)$-competitive algorithms, and a similar lower bound holds for online TSP, even on line metrics [35]. But the lower bound does not hold for FDP, which is *not* a generalization of TSP in the online case.

Another motivation for FDP is from the seemingly unrelated broadcast scheduling problem. In this problem, a server holds $n$ pages of varying sizes and requests are released over time, each of which is a query on one of the pages. The server can *broadcast* a page to all requests on that same page, which takes time equal to the page size. The objective is to minimize the maximum flow time. It is easy to reduce the broadcast scheduling problem to uncapacitated single-vehicle FDP on a star, where a page of size $s$ in the broadcast scheduling problem corresponds to an edge of length $s/2$ in the FDP problem.[1] It is known that FIFO gives $O(1)$-competitiveness for the broadcast scheduling problem [13, 11], and the single-vehicle uncapacitated FDP on stars [22]. One of our main results in the paper is an $O(1)$-competitive algorithm for uncapacitated FDP on *trees*, generalizing the results to tree metrics and the multiple-vehicle case. Mapping to the broadcast scheduling application, this gives the following more general setting. There is a tree rooted at $o$, where the pages correspond to the leaves of a tree, and each internal node corresponds to a setup procedure that takes some time to run. To broadcast a set of pages, in addition to the broadcasting time for each page, we have to spend time on running the setup procedures correspondent to the ancestor nodes of the pages.

## 1.1 Our Results

We state our results in this section. The formal definition of the offline and online food delivery problem (FDP) can be found in Section 2. In all the theorems below, $n$ is the number of points in the metric space.

[1] In FDP, we could define the completion time of a request as the time the vehicle returns to the depot after serving the request. The maximum flow time objective for the two versions differs by an $O(1)$ factor.

We first show that our problem also suffers from the maximum flow time objective, both in offline and online settings, even if we focus on the single-vehicle uncapacitated case.

▶ **Theorem 1.** *The following statements hold for the single-vehicle uncapacitated FDP:*

**(1a)** *It is NP-hard to approximate the offline problem within a factor of $o(n)$.*

**(1b)** *There is no $o(n)$-competitive algorithm for the online problem, even if it can run in exponential time and the metric is a bounded-pathwidth planar graph.*

**Our Results for Tree Metrics.** Given that (1b) holds even for bounded-pathwidth planar graphs, a natural candidate family of metrics is the tree metrics. Our first algorithmic result of the paper is that uncapacitated FDP on trees admit $O(1)$-competitive online algorithms:

▶ **Theorem 2** (Tree Metric)**.** *There's an efficient $O(1)$-competitive online algorithm for uncapacitated FDP on trees.*

The results in Theorem 1 and 2 are for uncapacitated version of the problem ($c = \infty$). We remark that these results should be contrasted with the lower bound given by Krumke et al.[35], which says that *online* TSP (and thus *uncapacitated* DaRP) does not admit $O(1)$-competitive algorithms *even on line metrics*. Their lower bound however does not apply to (online) uncapacitated FDP, and this is crucially due to the fact that in FDP a vehicle has to return to *the* depot before it can serve a newly released request.

We now turn to the general capacitated case. One can ask if the algorithm in Theorem 2 works for general $c$. Unfortunately, we show that this is impossible even for a very special case. Below $R$ is the set of requests in the instance:

▶ **Theorem 3.** *There is no $o(\sqrt{|R|})$-competitive online algorithm for single-vehicle FDP with $c = 2$ and on a tree with 5 vertices.*

**Our Results with Speed Augmentation.** Despite all the negative results, we show that the problem admits good competitive algorithms under the *speed augmentation* model (*in general metric*). In this model we allow vehicles in our algorithm to run faster than normal vehicles, while we compare the maximum flow time achieved by our algorithm using faster vehicles against the optimum maximum flow time that can be achieved with normal vehicles. The speeding factor defines how faster our vehicles are, and our goal is to decide the tradeoff between the speeding factor and the competitive (approximation) ratio of our algorithm. There are two reasons for which we consider this model. First, in the optimum solution of the hard instance for (1b), the vehicle is always busy. One mistake made by the online algorithm can delay the whole schedule. This situation rarely happens in practice, where one should expect the vehicles to have a reasonable amount of idle time. Second, the travel time between two points in practice is often an estimate and affected by many factors such as the driver's skill and the traffic condition. So, a solution that becomes very bad if the speeds of vehicles decrease slightly should be considered as very fragile. We remark that the speed augmentation model is also popular in many scheduling problems.

Let $\alpha_{\mathsf{TSP}}$ be the infinum of all values $\alpha$ such that there is a (polynomial time) $\alpha$-approximation algorithm for TSP. With the breakthrough result of Karlin, Klein, and Oveis Gharan [30], we know that $\alpha_{\mathsf{TSP}}$ is strictly smaller than 1.5. Define $\alpha_{\mathsf{CVRP}}$ similarly for the CVRP problem. It is known that $\alpha_{\mathsf{CVRP}} \le \alpha_{\mathsf{TSP}} + 1$. We give our results for the food delivery problem with speeding below.

▶ **Theorem 4** (General Metric). *For any small enough constant $\epsilon > 0$, there are*

**(4a)** *an exponential time $(1+\epsilon)$-speeding $O(1/\epsilon)$-competitive online algorithm for FDP,*

**(4b)** *a polynomial time $(\alpha_{\mathsf{TSP}} + \epsilon)$-speeding $O(1/\epsilon)$-competitive online algorithm for uncapacitated FDP, and*

**(4c)** *a polynomial time $(\alpha_{\mathsf{CVRP}} + \epsilon)$-speeding $O(1/\epsilon)$-competitive online algorithm for (capacitated) FDP.*

▶ **Theorem 5.** *The following statements hold.*

**(5a)** *There is NO $(1+\epsilon)$-speeding $o(1/\epsilon)$-competitive online algorithm for FDP.*

**(5b)** *For any constant $\alpha \in [1, \alpha_{\mathsf{TSP}})$, there is NO (polynomial time) $\alpha$-speeding $o(\sqrt{n})$-approximation algorithm for uncapaciated FDP.*

**(5c)** *For any constant $\alpha \in [1, \alpha_{\mathsf{CVRP}})$, there is NO (polynomial time) $\alpha$-speeding $o(\sqrt{n})$-approximation algorithm for (capaciated) FDP.*

Therefore, by (5a), the $O(1/\epsilon)$-dependence on $\epsilon$ in (4a) is needed. (5b) and (5c) state that the speeding factors in (4b) and (4c) are almost tight.

## 1.2 An Overview of Techniques

The hardness results in (1a), (5b) and (5c) are proved using simple reductions from TSP or CVRP, in which we repeat a TSP/CVRP instance multiple times. Since an efficient algorithm can not find the best TSP/CVRP tour, it needs to use a longer tour for each instance, which creates a delay in the schedule. The delays for all instances will accumulate, resulting in a bad flow time. The hardness remains if the speeding factor is not big enough.

The lower bounds in (1b), (5a) and Theorem 3 are proved using the same idea. We build a base instance satisfying the following property. At the beginning of the time horizon, the online algorithm needs to make a decision between two choices. If it made the incorrect choice, the total length of trips it uses to satisfy all requests will be 1 unit time longer than the case if it made the correct choice. The catch is, the algorithm will only know which choice is the correct one until near the end of the time horizon. So, it has to either wait for almost the whole time horizon, which will incur a large maximum flow time, or it will spend 1 more unit time on the trips, delaying the schedule. By repeating the base instance many times, the delay of the online algorithm will accumulate, resulting in a large maximum flow time. Lastly, we remark that although all such lower bounds are proved for deterministic algorithms, it is not hard to extend the proofs to randomized algorithms by directly applying Yao's Minimax Principle: instead of giving the correct choice adversarially, we make it uniformly random.

The overview of the algorithms for online uncapacitated FDP on trees will be given at the beginning of Section 3. The online algorithms in Theorem 4 are based on a simple idea. We wait for $\gamma F$ time units ($F$ is a given upper bound of the optimal flow time. See Section 2 for details.) for some $\gamma = \Theta\left(\frac{1}{\epsilon}\right)$. Then we know that the optimum solution can serve all the requests that arrived in the $\gamma F$-length interval using $(\gamma + O(1))F$ time. With $(1+\epsilon)$-speeding, the online algorithm can serve them in $\gamma F$ time. Thus, we can always guarantee an $O(F/\epsilon)$-flow time for all requests. If the algorithm has to be efficient, we need to lose a speeding factor of $\alpha_{\mathsf{TSP}} + \epsilon$ or $\alpha_{\mathsf{CVRP}} + \epsilon$, depending on whether the instance is uncapacitated.

## 1.3 Other related work

Most of previous results on CVRP and DaRP focus on the single vehicle case and the total travel distance objective, which is equivalent to the makespan. Unless specified otherwise, all the results surveyed below for CVRP and DaRP are for this case.

**Capacitated Vehicle Routing Problem (CVRP).** The CVRP is mostly studied in the offline setting. It admits an $(\alpha_{\mathsf{TSP}}+1)$-approximation, where $\alpha_{\mathsf{TSP}}$ is the approximation ratio for traveling salesman problem (TSP) [1, 24, 30]. A more general version of CVRP is studied in the literature: Each request has a demand, and we require the total demand of all requests satisfied in a single trip made by the vehicle is at most $c$. In the same papers, Altinkemer and Gavish [1] and Haimovich and Rinnooy Kan [24] gave an approximation algorithm with ratio $\alpha_{\mathsf{TSP}}+2$ for the general CVRP, which stood for over 30 years. Very recently, the approximation ratio for the problem has been improved to $\alpha_{\mathsf{TSP}}+2(1-\epsilon)$ by Blauth *et al.* [10], for some small constant $\epsilon>0$. There are also improved ratios on special metrics like Euclidean plane [16, 32], tree metrics [36, 44, 41], or other special graph metrics [15, 27].

One variant of CVRP that is related to the flow time objective is the *CVRP with bounded delay*: There is a *delay constraint* that any request $\rho$ spends at most $\beta d(r, v_\rho)$ time *on the vehicle*, where $v_\rho$ denotes the drop-off location of $\rho$. The goal is to find a minimum length route that satisfies all the delay constraints. For this variant, Gørtz *et al.* [21] gave a $\left(2.5+\frac{3}{\beta-1}\right)$-approximation algorithm for single-vehicle CVRP with bounded delay.

**Dial-a-Ride Problem (DaRP).** Like CVRP, the most studied setting for offline DaRP is also for the single-vehicle case with makespan objective, for which the best known algorithm achieves $\tilde{O}(\min\{\sqrt{n},\sqrt{c}\})$ approximation ratio [12, 23], where $n$ is the number of requests. The best known lower bound is only APX-hardness. The *online* DaRP is mostly studied with objectives like makespan [2] or total/weighted completion time [19, 34, 9]: all of these objectives admit small constant competitive-ratio algorithm. However, if we turn to minimizing the maximum flow-time, there exists no $o(n)$-competitive algorithm even on a 3-point uniform metric with a unit-capacity vechile [33].

A variant of DaRP studied in the literature that seems much easier is the preemptive DaRP, in which the vehicle is allowed to temporarily unload the cargo it carries in the middle of a trip, and pick it up later to resume delivery. The best approximation ratio for the problem with single vehicle is $O(\log n)$ [12], and there is a hardness of $\Omega(\log^{1/4} n)$ hardness of approximation [20]. For the multi-vehicle case, Gørtz *et al.*[21] gave an $O(\log^3 n)$ for the preemptive DaRP, and an $O(\log n)$-approximation when vehicles have infinite capacity.

**Flow-Time Scheduling.** The maximum flow time objective is studied in many scheduling problems. The simple FIFO strategy is known to achieve the best competitive ratio 2 [40] in the identical machine setting, and constant competitive algorithm exists even in related machine setting [6] (i.e., machines have different speed). However, the approximability changes dramatically if we switch the objective to the (weighted) sum of flow-time: if no preemption is allowed, the problem is $\Omega(m)$-hard in the online setting where $m$ is the number of machines, and $\Omega(m^{1/2-\epsilon})$-hard in the offline setting [31]. Even when preemption is allowed, offline $O(1)$-approximations (for the single-machine setting) are known only very recently [8, 18, 43], and there's an $\omega(1)$ lower bound for the online case [3].

For online broadcast scheduling, the maximum flow time objective also appears to be easier than other flow time objectives: max flow time admits 2-competitive algorithm [7, 11, 13], while average flow time has very strong lower bounds $\Omega(n)$ for deterministic algorithms [29] and $\Omega(\sqrt{n})$ for randomized algorithms [5].

**Resource Augmention in Flow-Time Scheduling.** Due to the strong lower bounds mentioned above for various flow time objectives, people proposed the *resource augmentation model* in order to find provably good algorithms. In a pioneering work, Kalyanasundaram and Pruhs [28] proposes the speed augmentation model where the algorithm is allowed to use machines faster than the optimal solution, and give the first constant competitive algorithm for minimizing (nonclairvoy) total flow time. Later Phillips *et al.*[42] explores another type of resource – machines: they showed that there is a constant competitive algorithm for non-preemptive weighted flow time scheduling, but uses $m + O(\log P)$ machines (here $P$ is the ratio of length between the longest and the shortest job, and the optimal solution uses only $m$ machines). They also showed that one can combine the two resources and get a constant-competitive algorithm for total flow time, using $O(\log n)$ extra machines with $O(1)$ speeding. Epstein and van Stee [17] gave a $\ell$-machine algorithm for the single-machine total flow time scheduling, and achieves an asymtotically optimal $O(\min\{\sqrt[\ell]{P}, \sqrt[\ell]{n}\})$-competitive ratio.

In the offline case, the resource augmentation model is also used to give constant approximation algorithms for *non-preemptive* flow time scheduling: first on a single machine [4], and then extended to multiple machines by Im *et al.*[26]. The resources are not equally powerful, though. Lucarelli *et al.*[39] showed that for the weighted flow time objective, the non-preemptive single-machine scheduling problem does not admit bounded-competitive algorithms *even with arbitrarily faster* machine. Choudhury *et al.*[14] therefore explored another type of "augmentation": allowing rejection of some jobs. This idea is used to obtain the first constant competitive algorithm for weighted flow time scheduling in the *non-preemptive* setting [39, 37, 38].

## 1.4 Organization

The remaining part of the paper is organized as follows. In Section 2, we formally define the offline and online food delivery problem. We prove our main algorithmic result, Theorem 2, by giving the $O(1)$-competitive algorithm for uncapacitated FDP on trees in Sections 3 and 4. The two sections consider the single-vehicle and multiple-vehicle cases respectively. Due to space limit, all lower bound results and the results with speed augmentation are included only in the full version.

## 2 Preliminary

We now define the food delivery problem formally, starting from the offline setting. We are given a graph $G = (V, E)$ with edge lengths $\ell : E \to \mathbb{R}_{>0}$, where $\ell(e)$ denotes the time needed to traverse the edge $e$ in either direction.[2] There is a special vertex $o \in V$ called the depot, which represents the restaurant. We are given a number $k \geq 1$ of vehicles which are initially located at $o$, and capacity $c \in \mathbb{Z}_{>0} \cup \{\infty\}$ on the vehicles. When $k = 1$, we say the problem is single-vehicle, and when $c = \infty$, we say the problem is uncapacitated. There is a set $R$ of requests. Each request $\rho \in R$ is denoted by $\rho = (r_\rho, v_\rho)$, where $r_\rho \in \mathbb{R}_{\geq 0}$ and $v_\rho \in V$ are the arrival time and the delivery location of the request respectively.

To describe the output of the problem, we need to define *trips*. A trip is defined by a triple $(t, (u_0 = o, u_1, u_2, \cdots, u_z = o), R')$, where $t \geq 0$ is the starting time of the trip, $(u_0 = o, u_1, u_2, u_3, \cdots, u_{z-1}, u_z = o)$ is a (possibly complex) cycle in $G$ that starts and ends at $o$, and $R' \subseteq R$, $|R'| \leq c$, is the set of requests served by the trip. So $z \geq 2$ and $(u_{z'-1}, u_{z'}) \in E$ for every $z' \in [z]$. We require the following properties to hold for a trip.

[2] Equivalently we could use a metric $(V, d)$ to describe the travel times, but for many of our results it is more convenient to use the graph $G$ with edge lengths.

First, $u_{z'} \neq o$ for every $z' \in [z-1]$; one can see easily soon that this is without loss of generality. Then, for every served request $\rho \in R'$, we have $r_\rho \leq t$ and $v_\rho$ appears in the cycle. If $\tilde{z}$ is the smallest index such that $u_{\tilde{z}} = v_\rho$, then we say $\rho$ is served by the trip at time $t + \sum_{z'=1}^{\tilde{z}} \ell(u_{z'-1}, u_{z'})$. The completion time of the trip is defined as $t + \sum_{z'=1}^{z} \ell(u_{z'-1}, u_{z'})$. Abusing the definition slightly, sometimes we also use the word "trip" to denote the cycle $(u_0 = o, u_1, u_2, \cdots, u_z = o)$, with $t$ and $R'$ specified separately.

The output of the food delivery problem contains $k$ sequences of trips correspondent to the itineraries of the $k$ vehicles. The following properties need to be satisfied. For every pair of adjacent trips in any of the $k$ sequences, the starting time of the latter trip is at least the completion time of the former one. Moreover, each request in $R$ is served by exactly one trip in the $k$ sequences; in other words, the sets of served requests in all trips of the $k$ sequences form a partition of $R$. Let $t_\rho$ be the time that a request $\rho \in R$ is served (by the unique trip that serves it). We define the flow time of $\rho$ to be $t_\rho - r_\rho$. The goal of the problem is to minimize the maximum flow time, i.e, $\max_{\rho\in R}(t_\rho - r_\rho)$.

So far we have defined the food delivery problem in the offline setting. In the online setting, $G, \ell, k$ and $c$ are given upfront, but the requests arrive online. Once we decided to start a trip at time $t$, then we have to complete the trip as planed. Formally, we require that for any time $t \geq 0$, the prefixes of the $k$ sequences of trips with starting time at or before $t$ can only depend on the requests that arrive at or before $t$.

**Guessing the Optimum Maximum Flow Time Online.** When designing online algorithms, we shall use the standard doubling trick to assume that we are given an upper bound $F$ on the optimum maximum flow time of the instance, and the competitive ratio is defined by comparing to $F$. That is, a $\beta$-competitive online algorithm has maximum flow time at most $\beta F$. If we have a $\beta$-competitive online algorithm $\mathcal{A}$ under this setting, then we can obtain an $8\beta$-competitive algorithm $\mathcal{A}'$ under the setting where $F$ is not given to us, while keeping the speeding factor and running time unchanged. In the paper, we are not trying to optimize the constant, and the factor of 8 will be hidden in the $O(\cdot)$ notation. Due to space limitation, we omit the detail here.

## 3 Online Uncapacitated Single-Vehicle FDP on Tree Metrics

In this section and the next one, we give the $O(1)$-competitive algorithm for online uncapacitated FDP on tree metrics, proving Theorem 2. In this section, we consider the case $k = 1$, i.e, there is only one vehicle. We separate this case from the general problem as its algorithm is simpler and the competitive ratio we obtain is better.

First we setup some notations here. Let $T = (V, E)$ be the tree and we assume it is rooted at the depot $o \in V$. For every $v \in V$, we use $V_v$ to denote the set of descendants of $v$ (including $v$ itself). For an edge $e = (u, v)$ with $v$ being the child, we define $V_e = V_v$. We use $d$ to denote the metric induced by the tree $T$ with lengths $\ell(\cdot)$. Given a set $X \subseteq V$ of vertices, we define $\mathsf{mst}(X)$ to be the cost of the minimal sub-tree of $T$ containing $X$ and $o$ (Notice that we require the tree to contain $o$). Suppose we are further given an element $e \in V \cup E$. We define $\mathsf{mst}_e(X) = \mathsf{mst}(V_e \cap X)$, which is the cost of the minimal sub-tree of $T$ containing $o$ and all vertices in $V_e \cap X$. So if $V_e \cap X = \emptyset$, then $\mathsf{mst}_e(X) = 0$; otherwise, the tree contains all the edges from $o$ to $e$ (including $e$ itself if it is an edge). By definition we have $\mathsf{mst}_o(X) = \mathsf{mst}(X)$.

Since most of the time we deal with requests, it is convenient for us to use $\mathsf{mst}(R')$ and $\mathsf{mst}_e(R')$ to denote $\mathsf{mst}(\{v_\rho : \rho \in R'\})$ and $\mathsf{mst}_e(\{v_\rho : \rho \in R'\})$ respectively for a set $R' \subseteq R$ of requests. Abusing notations slightly, sometimes we also use $\mathsf{mst}(X)$ to denote the actual sub-tree of $T$ achieving the cost $\mathsf{mst}(X)$. This also extends to $\mathsf{mst}_e(X)$, $\mathsf{mst}(R')$ and $\mathsf{mst}_e(R')$.

We now overview the algorithm for the online uncapacitated FDP problem on tree metrics, for both the single and multiple-machine cases. We break the time horizon into intervals of length $F$: $\{[0,F),[F,2F),[2F,3F),[3F,4F),...\}$ and index them by $1,2,3,4,\cdots$. Recall that $F$ is an upper bound on the optimum maximum flow time, and the competitive ratio of our algorithm is defined against $F$. Let $R_i$ be the set of requests that arrive in interval $i$. For each even integer $i$, we break $R_i$ into $R_i^{\mathsf{left}}$ and $R_i^{\mathsf{right}}$, merge $R_i^{\mathsf{left}}$ with $R_{i-1}$, merge $R_i^{\mathsf{right}}$ with $R_{i+1}$, so as to minimize some carefully designed objective. So for each odd integer $i$, we have constructed a merged set $R'_i := R_{i-1}^{\mathsf{right}} \cup R_i \cup R_{i+1}^{\mathsf{left}}$, which we call a *bundle*. The bundles then can be constructed in an online manner: the bundle $R'_i$ is determined by requests that arrive before time $(i+2)F$.

Then our online algorithm handles the bundles separately. When $k=1$, we view each bundle $R'_i$ as a job of size $2\mathsf{mst}(R'_i)$; when $k \geq 2$, we break $R'_i$ into many *groups* and treat each group $Q$ as a job of size $2\mathsf{mst}(Q)$. We think of all these jobs are released at time $(i+2)F$. We then assign the jobs to the vehicles online in a greedy manner. A crucial lemma we show is that the jobs have a small backlog: The jobs released in any interval $[aF,bF]$ have total size at most $(b-a)F + O(1)\cdot kF$, and each job has size $O(F)$. The properties guarantee that all the jobs complete with an $O(F)$ flow time.

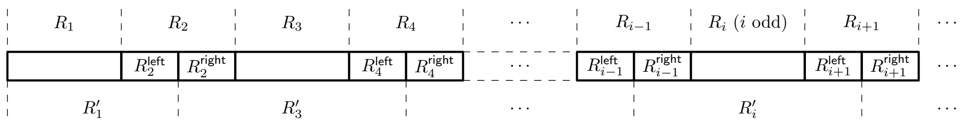
## 3.1 Description of Algorithm for Single-Vehicle ($k = 1$) Case

Now we formally state the algorithm for the case $k=1$. For every integer $i \geq 1$, let $R_i = \{\rho \in R : (i-1)F \leq r_\rho < iF\}$ as stated. Indeed, once we know a request is in $R_i$, we do not care about its precise arrival time anymore. In the first step of the online algorithm, we partition the requests into bundles $(R'_i)_{i\geq 1: i \text{ is odd}}$, using Algorithm 1, where the bundle $R'_i$ is generated at time $(i+2)F$. For now let us focus on the case $k=1$ in the algorithm. The formal definition for $\mathsf{cost}(\cdot|\cdot)$ is postponed to section 4 since it's not used here. Throughout the section and the next one, we assume all undefined sets are set to $\emptyset$. See Figure 1(a) for an illustration of the relationships between $R_i$, $R'_i$, $R_i^{\mathsf{left}}$ and $R_i^{\mathsf{right}}$.

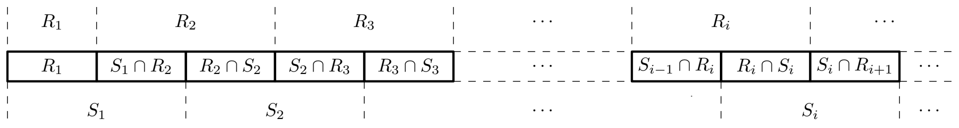
**Algorithm 1** Partition of Requests into Bundles for Both Single and Multiple-Vehicle Cases.

```
1: for $i = 2, 4, 6, 8, \cdots$ do
2:     wait until time $(i+1)F$
3:     let $R_i = \{\rho \in R : r_\rho \in [(i-1)F, iF)\}$ be as defined in the text
4:     partition $R_i$ into $R_i^{\mathsf{left}}$ and $R_i^{\mathsf{right}}$ so as to minimize
           $\begin{cases} \mathsf{mst}(R_{i-1} \cup R_i^{\mathsf{left}}) + \mathsf{mst}(R_i^{\mathsf{right}} \cup R_{i+1}) & \text{if } k = 1 \\ \mathsf{cost}(R_i^{\mathsf{left}} | R_{i-1}) + \mathsf{cost}(R_i^{\mathsf{right}} | R_{i+1}) & \text{if } k > 1 \end{cases}$
5:     release the bundle $R'_{i-1} := R_{i-2}^{\mathsf{right}} \cup R_{i-1} \cup R_i^{\mathsf{left}}$ (at time $(i+1)F$)
```

We briefly talk about the efficiency of the algorithm for $k=1$. It is easy to see that the following simple strategy will find the partition $(R_i^{\mathsf{left}}, R_i^{\mathsf{right}})$ that minimizes $\mathsf{mst}(R_{i-1} \cup R_i^{\mathsf{left}}) + \mathsf{mst}(R_i^{\mathsf{right}} \cup R_{i+1})$. For every $\rho \in R_i$, if $d(\rho, \mathsf{mst}(R_{i-1})) \leq d(\rho, \mathsf{mst}(R_{i+1}))$, then we put $\rho$ in $R_i^{\mathsf{left}}$. Otherwise we put $\rho$ in $R_i^{\mathsf{right}}$. Here $d(\rho, \mathsf{mst}(R_{i-1}))$ ($d(\rho, \mathsf{mst}(R_{i+1}))$, resp.) is the shortest distance between $r_\rho$ and any vertex in $\mathsf{mst}(R_{i-1})$ ($\mathsf{mst}(R_{i+1})$, resp.).



**(a)** Illustration of relationships between $R_i$'s, $R_i^{\text{left}}$'s, $R_i^{\text{right}}$'s and $R'_i$'s. For every even $i$, we have $R_i = R_i^{\text{left}} \cup R_i^{\text{right}}$. For every odd $i$, we have $R'_i = R_{i-1}^{\text{right}} \cup R_i \cup R_{i+1}^{\text{left}}$.



**(b)** Illustration of relationships between $R_i$'s and $S_i$'s. For every $i$, we have $R_i \subseteq S_{i-1} \cup S_i$ and $S_i \subseteq R_i \cup R_{i+1}$.

**Figure 1** Illustration of sets used in the algorithm and analysis for online uncapacitated FDP on tree metrics.

Notice that starting from the depot $o$, serving all the requests in $R'_i$ and traveling back to the depot take time exactly $2\mathsf{mst}(R'_i)$. Therefore, we can view each bundle $R'_i$, for an odd $i \geq 1$, as a job of size $2\mathsf{mst}(R'_i)$ released at time $(i+2)F$. We view the vehicle as a machine. In the second step of the online algorithm, we then schedule the jobs (i.e., bundles) on the machine (i.e., the vehicle) in their order of releasing: Whenever the machine is idle and there are available jobs, we process the job that is released the earliest. This finishes the description of the algorithm for the single vehicle case.

## 3.2 Analysis of Backlog of Jobs

We define the flow time of a job $R'_i$, for an odd $i \geq 1$, as its completion time minus its release time (which is $(i+2)F$). By the folklore result, if the total size of all jobs released in time $[t, t']$ is at most $t' - t + D$ for some $D$ and for every pair $t \leq t'$ of time points, then every job has flow time at most $D$ in the schedule using greedy algorithm. Therefore, it remains to prove that $D$ is small, which is stated in Lemma 9 later.

Before describing and proving the lemma, we introduce a partition $(S_i)_{i\geq 1}$ of requests that depends on the offline optimum solution, and prove some helper lemmas. Notice that in the offline optimum solution a trip can not serve two requests whose arrival times are more than $F$ apart: If that happens, then the flow time of the earlier request of the two is more than $F$. Thus, a trip in the optimum solution serves either requests from a single set $R_i$ for some $i$, or requests from two sets $R_i$ and $R_{i+1}$ for some $i$. In both the cases, we put all the requests in the trip into the set $S_i$. Therefore, $S_1, S_2, S_3, \cdots$ form a partition of $R$. Notice that $S_i \subseteq R_i \cup R_{i+1}$ and $R_i \subseteq S_{i-1} \cup S_i$ for every integer $i$. See Figure 1(b) for an illustration of the relationship between $R_i$'s and $S_i$'s. Notice that $S_i$'s is only used in our analysis, as they depend on the offline optimum solution, which our algorithm does not know.

We need the following simple lemma and corollaries.

▶ **Lemma 6.** *Let $X_1, X_2, X_3$ and $X_4$ be subsets of $V$ and $e \in E \cup V$. Then the following inequalities hold:*

$$\mathsf{mst}_e(X_1 \cup X_2) \leq \mathsf{mst}_e(X_1) + \mathsf{mst}_e(X_2), \tag{1}$$

$$\mathsf{mst}_e(X_2) + \mathsf{mst}_e(X_1 \cup X_2 \cup X_3) \leq \mathsf{mst}_e(X_1 \cup X_2) + \mathsf{mst}_e(X_2 \cup X_3), \tag{2}$$

$$\mathsf{mst}_e(X_1 \cup X_2 \cup X_3) + \mathsf{mst}_e(X_2 \cup X_3 \cup X_4) \leq \mathsf{mst}_e(X_1 \cup X_2) + \mathsf{mst}_e(X_2 \cup X_3) + \mathsf{mst}_e(X_3 \cup X_4). \tag{3}$$

**Proof.** It suffices to prove the inequalities for $e = o$, since the other cases can be proved by changing $X_1, X_2, X_3$ and $X_4$ to $V_e \cap X_1, V_e \cap X_2, V_e \cap X_3$ and $V_e \cap X_4$ respectively.

(1) is easy to see. For (2), we focus on each edge $e' \in E$ and count the number of times $\ell(e')$ is considered on the left and right sides respectively. Notice that $\ell(e')$ is counted in $\mathsf{mst}(Y)$ for some set $Y$ if and only if $V_{e'} \cap Y \neq \emptyset$.

- If $\ell(e')$ is counted 0 times on the left side, then $V_{e'} \cap Y = \emptyset$ for every $Y \in \{X_1, X_2, X_3\}$. Clearly $\ell(e')$ is counted 0 times on the right side.
- If $\ell(e')$ is counted in $\mathsf{mst}(X_1 \cup X_2 \cup X_3)$ but not in $\mathsf{mst}(X_2)$, then either $V_{e'} \cap X_1 \neq \emptyset$ or $V_{e'} \cap X_3 \neq \emptyset$. Then $\ell(e')$ is counted at least once on the right side.
- If $\ell(e')$ is counted in $\mathsf{mst}(X_2)$, then it is counted twice on both the left right sides.

Similarly for the proof of (3), we consider the number of times each $\ell(e')$ is counted on both sides:

- If $e'$ is counted 0 times on the left side, then $V_{e'} \cap Y = \emptyset$ for every $Y \in \{X_1, X_2, X_3, X_4\}$. Thus, $\ell(e')$ is counted 0 times on the right side.
- Assume $\ell(e')$ is counted once on the left side. W.l.o.g assume it is counted in $\mathsf{mst}(X_1 \cup X_2 \cup X_3)$ but not in $\mathsf{mst}(X_2 \cup X_3 \cup X_4)$. Then clearly $e'$ is counted at least once on the right side.
- Finally assume $\ell(e')$ is counted in both $\mathsf{mst}(X_1 \cup X_2 \cup X_3)$ and $\mathsf{mst}(X_2 \cup X_3 \cup X_4)$. If $V_{e'} \cap X_2 \neq \emptyset$ or $V_{e'} \cap X_3 \neq \emptyset$, then $\ell(e')$ is counted at least twice on the right side. Otherwise we have $V_{e'} \cap X_1 \neq \emptyset$ and $V_{e'} \cap X_4 \neq \emptyset$. In this case $\ell(e')$ is counted twice on the right side. ◀

▶ **Corollary 7.** *For every odd $i \geq 1$ and $e \in E \cup V$, we have $\mathsf{mst}_e(R'_i) \leq \mathsf{mst}_e(R^{\mathsf{right}}_{i-1} \cup R_i) + \mathsf{mst}_e(R_i \cup R^{\mathsf{left}}_{i+1}) - \mathsf{mst}_e(R_i)$.*

**Proof.** Applying (2) with $X_1 = R^{\mathsf{right}}_{i-1}, X_2 = R_i$ and $X_3 = R^{\mathsf{left}}_{i+1}$, and using $X_1 \cup X_2 \cup X_3 = R'_i$ proves the lemma. [3] ◀

▶ **Corollary 8.** *For every integer $i \geq 1$ and $e \in E \cup V$, we have $\mathsf{mst}_e(S_{i-1} \cup R_i) + \mathsf{mst}_e(R_i \cup S_i) - \mathsf{mst}_e(R_i) \leq \mathsf{mst}_e(S_{i-1}) + \mathsf{mst}_e(S_i)$.*

**Proof.** We apply (3) with $X_1 = R_{i-1} \cap S_{i-1}, X_2 = S_{i-1} \cap R_i, X_3 = R_i \cap S_i$ and $X_4 = S_i \cap R_{i+1}$. The corollary follows by that $X_1 \cup X_2 = S_{i-1}$, $X_2 \cup X_3 = R_i$, $X_3 \cup X_4 = S_i$, $X_1 \cup X_2 \cup X_3 = S_{i-1} \cup R_i$ and $X_2 \cup X_3 \cup X_4 = R_i \cup S_i$. ◀

With the corollaries established, we now state and prove the lemma that bounds the backlog of bundles.

▶ **Lemma 9.** *For any two odd positive integers $a \leq b$, we have $2\sum_{i \in [a,b]: i \text{ } odd} \mathsf{mst}(R'_i) \leq (b - a + 4)F$.*

**Proof.**

$$\begin{aligned} & \sum_{i \in [a,b]: i \text{ odd}} \mathsf{mst}(R'_i) \\ \leq & \sum_{i \in [a,b]: i \text{ odd}} \left( \mathsf{mst}(R^{\mathsf{right}}_{i-1} \cup R_i) + \mathsf{mst}(R_i \cup R^{\mathsf{left}}_{i+1}) - \mathsf{mst}(R_i) \right) \qquad \text{(by Corollary 7)} \end{aligned}$$

[3] Recall that in the notation $\mathsf{mst}_e$, we can view requests as vertices by ignoring their arrival times.

$$\begin{aligned}
&= \sum_{i\in(a,b):i \text{ even}} \Big(\mathsf{mst}(R_{i-1}\cup R_i^{\mathsf{left}}) + \mathsf{mst}(R_i^{\mathsf{right}}\cup R_{i+1})\Big) - \sum_{i\in[a,b]:i \text{ odd}} \mathsf{mst}(R_i)\\
&\quad + \mathsf{mst}(R_{a-1}^{\mathsf{right}}\cup R_a) + \mathsf{mst}(R_b\cup R_{b+1}^{\mathsf{left}}) \qquad \text{(by reorganizing terms)}\\
&\le \sum_{i\in(a,b):i \text{ even}} \Big(\mathsf{mst}(R_{i-1}\cup S_{i-1}) + \mathsf{mst}(S_i\cup R_{i+1})\Big) - \sum_{i\in[a,b]:i \text{ odd}} \mathsf{mst}(R_i)\\
&\quad + \mathsf{mst}(S_{a-2}\cap R_{a-1}) + \mathsf{mst}(S_{a-1}\cup R_a) + \mathsf{mst}(R_b\cup S_b) + \mathsf{mst}(R_{b+1}\cap S_{b+1}) \qquad (4)\\
&= \sum_{i\in[a,b]:i \text{ odd}} \big(\mathsf{mst}(S_{i-1}\cup R_i) + \mathsf{mst}(R_i\cup S_i) - \mathsf{mst}(R_i)\big)\\
&\quad + \mathsf{mst}(S_{a-2}\cap R_{a-1}) + \mathsf{mst}(R_{b+1}\cap S_{b+1}) \qquad \text{(by reorganizing terms)}\\
&\le \sum_{i\in[a,b]:i \text{ odd}} \big(\mathsf{mst}(S_{i-1}) + \mathsf{mst}(S_i)\big) + \mathsf{mst}(S_{a-2}\cap R_{a-1}) + \mathsf{mst}(R_{b+1}\cap S_{b+1})\\
&\qquad \text{(by Corollary 8)}\\
&= \mathsf{mst}(S_{a-2}\cap R_{a-1}) + \sum_{i=a-1}^{b} \mathsf{mst}(S_i) + \mathsf{mst}(R_{b+1}\cap S_{b+1}).
\end{aligned}$$

It remains to argue about (4). Notice that $S_{i-1}\cap R_i$ and $R_i\cap S_i$ form a partition of $R_i$. By the way we obtain $R_i^{\mathsf{left}}$ and $R_i^{\mathsf{right}}$, we have $\mathsf{mst}(R_{i-1}\cup R_i^{\mathsf{left}}) + \mathsf{mst}(R_i^{\mathsf{right}}\cup R_{i+1}) \le \mathsf{mst}(R_{i-1}\cup(S_{i-1}\cap R_i)) + \mathsf{mst}((R_i\cap S_i)\cup R_{i+1}) = \mathsf{mst}(R_{i-1}\cup S_{i-1}) + \mathsf{mst}(S_i\cup R_{i+1})$. Then $R_{a-1}^{\mathsf{right}}\cup R_a \subseteq R_{a-1}\cup R_a = (S_{a-2}\cap R_{a-1})\cup S_{a-1}\cup R_a$ and $R_b\cup R_{b+1}^{\mathsf{left}} \subseteq R_b\cup R_{b+1} = R_b\cup S_b\cup(R_{b+1}\cap S_{b+1})$. Combining the facts with (1) gives (4).

Now we prove that $2\Big(\mathsf{mst}(S_{a-2}\cap R_{a-1}) + \sum_{i=a-1}^{b}\mathsf{mst}(S_i) + \mathsf{mst}(R_{b+1}\cap S_{b+1})\Big) \le (b-a+4)F$, which finishes the proof of the lemma. We define $\mathcal{S} = \{(S_{a-2}\cap R_{a-1}), S_{a-1}, S_a, S_{a+1}, \cdots, S_b, (R_{b+1}\cap S_{b+1})\}$ for convenience. Then $\mathcal{S}$ forms a partition of $Q := R_{a-1}\cup R_a\cup R_{a+1}\cup\cdots\cup R_{b+1}$. Focus on the trips in the optimum solution that serve at least one request in $Q$. By the definition of $S_i$'s, each such trip can not serve requests from two different sets in $\mathcal{S}$. Moreover, such a trip can not start before time $(a-2)F$ since all requests in $Q$ arrive no earlier than $(a-2)F$. It can not end after $(b+1)F+F=(b+2)F$ since all requests in $Q$ arrive before $(b+1)F$ and the maximum flow time of the optimum solution is at most $F$. Therefore, $2\sum_{S\in\mathcal{S}}\mathsf{mst}(S)$ is at most the total length of these trips, which is at most $(b+2)F-(a-2)F=(b-a+4)F$. ◀

By Lemma 9, the total size of jobs (i.e, bundles) released in $[t,t']$ is at most $(t'-t)+4F$ for any $t\le t'$, using the language of the scheduling setting. Therefore, the greedy scheduling algorithm produces a schedule with maximum flow time at most $4F$. For an odd $i$, $R'_i$ is released at time $(i+2)F$, and thus all requests in $R'_i$ are served by time $(i+6)F$. Since requests in $R'_i$ arrive no earlier than $(i-2)F$, the maximum flow time of all requests is at most $8F$. This finishes the proof of Theorem 2 for the case $k=1$.

## 4 Online Uncapacitated Multiple-Vehicle FDP on Tree Metrics

Now we move on to the case of general $k$ for online FDP on trees. The first step is, again, using Algorithm 1 to partition jobs into bundles. As we mentioned, the main differences between the multi-vehicle case and the single-vehicle case are: We use a different criteria to break $R_i$ for an even $i\ge 2$ into $R_i^{\mathsf{left}}$ and $R_i^{\mathsf{right}}$, and we break a bundle $R'_i$ for an odd $i\ge 1$ into groups and treat each group as a job (instead of treating the whole bundle as a job).

## 4.1 Partitioning $R$ into Bundles

As before we also generate the set $(R'_i)_{i \geq 1, i \text{ is odd}}$ in the first step, which is also described in Algorithm 1. To break $R_i$ into $R_i^{\mathsf{left}}$ and $R_i^{\mathsf{right}}$, we use the function $\mathsf{cost}(R_i^{\mathsf{left}}|R_{i-1}) + \mathsf{cost}(R_i^{\mathsf{right}}|R_{i+1})$. We define the relevant notations now.

Given a real number $F' > 0$ and a set $X$ of requests, we define

$$c_{F'}(X, e) = \left\lceil \frac{\mathsf{mst}_e(X)}{F'} \right\rceil, \forall e \in E, \text{ and } \mathsf{cost}_{F'}(X) = 2\sum_{e \in E} c_{F'}(X, e)\ell(e).$$

Most of the time we shall use the definitions with $F' = F$. Therefore we simply use $c(X, e)$ and $\mathsf{cost}(X)$ to denote $c_F(X, e)$ and $\mathsf{cost}(X)$.

Then, for two sets $X$ and $X'$ of requests, we define

$$c(X, e|X') = \begin{cases} \left\lceil \frac{\mathsf{mst}_e(X)}{F} \right\rceil = c(X, e) & \text{if } V_e \cap X' = \emptyset \\ \left\lfloor \frac{\mathsf{mst}_e(X' \cup X) - \mathsf{mst}_e(X')}{F} \right\rfloor & \text{if } V_e \cap X' \neq \emptyset \end{cases}, \quad \forall e \in E,$$

$$\text{and } \mathsf{cost}(X|X') = 2\sum_{e \in E} c(X, e|X')\ell(e).$$

By the definition, we have $c(X, e|X') = c(X \setminus X', e|X')$, and $\mathsf{cost}(X|X') = \mathsf{cost}(X \setminus X'|X')$.

We now elaborate more on the definitions. Unlike the single-vehicle case, we need to break each bundle $R'_i$ created into many groups, to make sure that each group can be served in time $O(F)$. Then an edge $e \in E$ in $\mathsf{mst}(R'_i)$ may need to be used by many trips to satisfy the property. Then $c(X, e)$ gives a lower bound on the number of bi-directional traverses of $e$ needed to serve $X$ in the offline optimum solution. Thus $\mathsf{cost}(X)$ gives the total time needed to serve $X$. Notice if a trip uses an edge $e$, it uses $e$ twice, hence the factor of 2.

For $c(X, e|X')$, one can think of it as $c(X \cup X', e) - c(X', e) = \left\lceil \frac{\mathsf{mst}_e(X \cup X')}{F} \right\rceil - \left\lceil \frac{\mathsf{mst}_e(X')}{F} \right\rceil$, which is at least $\left\lfloor \frac{\mathsf{mst}_e(X \cup X')}{F} - \frac{\mathsf{mst}_e(X')}{F} \right\rfloor$. That is, the extra number of traverses of $e$ needed if we grow the set of request positions from $X'$ to $X \cup X'$. In the actual definition, we use the lower bound instead if $V_e \cap X' \neq \emptyset$.

## 4.2 Upper Bound on Costs of Bundles

The main goal of the section is to prove the following theorem:

▶ **Theorem 10.** *For every two odd positive integers $a \leq b$, we have $\sum_{i \in [a,b]} \mathsf{cost}_{3F}(R'_i) \leq k(b - a + 4)F$.*

We prove the following three inequalities, which will imply the theorem:

$$\sum_{i \in [a,b]} \mathsf{cost}_{3F}(R'_i) \leq \sum_{i \in [a,b]: i \text{ odd}} \Big(\mathsf{cost}(R_i) + \mathsf{cost}(R_{i-1}^{\mathsf{right}}|R_i) + \mathsf{cost}(R_{i+1}^{\mathsf{left}}|R_i)\Big) \tag{5}$$

$$\leq \mathsf{cost}(S_{a-2} \cap R_{a-1}) + \sum_{i=a-1}^{b} \mathsf{cost}(S_i) + \mathsf{cost}(R_{b+1} \cap S_{b+1}) \tag{6}$$

$$\leq k(b - a + 4)F. \tag{7}$$

We first prove (7), by showing that the $\mathsf{cost}()$ function indeed capture the length of trips in optimum solution.

▷ Claim 11. Let $R' \subseteq R$, and $\mathcal{P}$ be the set of trips in the offline optimum solution that serve at least one request in $R'$. Then every $e \in E$ is used by at least $c(R', e)$ trips in $\mathcal{P}$, implying that the total cost of $\mathcal{P}$ is at least $\mathsf{cost}(R')$.

Proof. Focus on each edge $e \in E$, and assume $e = (u, v)$, where $v$ is the child vertex. If $V_e \cap R' = \emptyset$ then $c(R', e) = 0$ and the statement holds trivially. If $V_e \cap R' \neq \emptyset$ and $\mathsf{mst}_e(R') \leq F$, then $c(R', e) = 1$ and at least 1 trip in $\mathcal{P}$ uses $e$ and the claim also holds. So, from now on, we assume $\mathsf{mst}_e(R') > F$.

Focus on a trip $P \in \mathcal{P}$ that uses $e$. For convenience, we treat $P$ as the sub-tree of edges used by $P$, without double-counting each edge. The total length of descendant edges of $e$ in $P$ (excluding $e$ itself) is at most $F - d(o, v)$. This holds since $P$ contains edges with a total length of at most $F$, and it contains the edges from $o$ to $v$. On the other hand, all the descendant edges of $v$ in $\mathsf{mst}_e(R')$ should be contained in $\mathcal{P}$. The total length of these edges is $\mathsf{mst}_e(R') - d(o, v)$. Therefore, the number of trips that use $e$ is at least $\left\lceil \frac{\mathsf{mst}_e(R') - d(o,v)}{F - d(o,v)} \right\rceil \geq \left\lceil \frac{\mathsf{mst}_e(R')}{F} \right\rceil = c(R', e)$, where the inequality comes from that $\mathsf{mst}_e(R') > F \geq d(o, v)$. The lemma holds as each trip that traverses $e$ does this twice. ◁

**Proof of** (7)**.** The argument is similar to that in the last paragraph inside the proof of Lemma 9, except now we use $\mathsf{cost}(S)$, instead of $2\mathsf{mst}(S)$, to lower bound the length of trips for a set $S \in \mathcal{S}$ of requests, and there are $k \geq 2$ vehicles. ◀

Then we turn to (6). We first show some simple inequalities about $\mathsf{cost}$ function.

▶ **Lemma 12.** *For any $X_1, X_2, X_3, X_4 \subseteq V$, we have*

$$\mathsf{cost}(X_1 \cup X_2 | X_3) \leq \mathsf{cost}(X_1) + \mathsf{cost}(X_2 | X_3), \tag{8}$$

$$\mathsf{cost}(X_2 \cup X_3) + \mathsf{cost}(X_1 | X_2 \cup X_3) + \mathsf{cost}(X_4 | X_2 \cup X_3) \leq \mathsf{cost}(X_1 \cup X_2) + \mathsf{cost}(X_3 \cup X_4). \tag{9}$$

Let $S_i$'s be defined in the same way as in the single-vehicle case: all the requests in a trip containing only requests from $R_i$, or requests from $R_i$ and $R_{i+1}$, are put into $S_i$.

▶ **Corollary 13.** *For any $i$, we have* $\mathsf{cost}(R_i) + \mathsf{cost}(R_{i-1} \cap S_{i-1} | R_i) + \mathsf{cost}(S_i \cap R_{i+1} | R_i) \leq \mathsf{cost}(S_{i-1}) + \mathsf{cost}(S_i)$.

**Proof.** The corollary follows from (9) by setting $X_1 = R_{i-1} \cap S_{i-1}, X_2 = S_{i-1} \cap R_i, X_3 = R_i \cap S_i$ and $X_4 = S_i \cap R_{i+1}$. ◀

Recall that in our algorithm, for every even $i$, we break $R_i$ into $R_i^{\mathsf{left}}$ and $R_i^{\mathsf{right}}$ so as to minimize $\mathsf{cost}(R_i^{\mathsf{left}} | R_{i-1}) + \mathsf{cost}(R_i^{\mathsf{right}} | R_{i+1})$. Then, we can proceed to show (6) and (5), whose proof are tedious but straightforward. Due to space limit, we leave the complete calculation in the full version.

## 4.3 Breaking Bundles into Groups

In this section, we break each bundle $R'_i$ into many groups $\mathcal{Q}_i$ as in the following lemma.

▶ **Lemma 14.** *For every odd integer $i \geq 1$, we can efficiently find a partition $\mathcal{Q}_i$ of $R'_i$ such that $\mathsf{mst}(Q) \leq 8F$ for every $Q \in \mathcal{Q}_i$, and $2\sum_{Q \in \mathcal{Q}_i} \mathsf{mst}(Q) \leq \mathsf{cost}_{3F}(R'_i)$.*

**Proof.** Within this proof, we need to change $T$ to a binary tree by replacing each internal vertex $v$ with at least three children with a binary tree. For every such vertex $v$ with $d_v \geq 3$ children, we replace the star containing $v$ and its children by a gadget, which is a complete binary tree with $d_v$ leaves. We then identify the root of the gadget with $v$, and the leaves with the $d_v$ children of $v$. In the gadget, the length of an edge incident to a child $u$ of $v$ is set to $\ell(u,v)$, and the length of an edge not incident to a child is set to 0. It is easy to see that this transformation does not change the instance, except now we have edges of length 0. After this step, every internal vertex of $T$ has degree exactly 2.

For any vertex $v$ in $T$ and a set $R'$ of requests, we define $\mathsf{mst}'_v(R')$ to be cost of the minimum spanning tree containing $v$ and $V_v \cap R'$, where $V_v$ is the set of descendant vertices of $v$ in $T$ (including $v$ itself). Notice an important difference between the definition of $\mathsf{mst}'_v(R')$ and $\mathsf{mst}_v(R')$ for an edge $e$: for $\mathsf{mst}'_v(R')$ the tree does not need to contain the depot $o$ and thus the quantity does not count the total length $d(o,v)$ of edges from $o$ to $v$. Similarly, for an edge $e=(u,v)$ with $v$ being the child end-vertex, we use $\mathsf{mst}'_e(R')$ to denote $\mathsf{mst}'_v(R')$. Also, sometimes we use $\mathsf{mst}'_v(R')$ to denote the tree achieving the cost $\mathsf{mst}'_v(R')$.

The following is the pseudo-code for constructing $\mathcal{Q}_i$:

```
1: $R' \leftarrow R'_i$, $\mathcal{Q}_i \leftarrow \emptyset$.
2: while $R' \neq \emptyset$ do
3:     choose a lowest vertex $v$ such that $\mathsf{mst}'_v(R') \geq 3F$; if no vertices $v$ satisfy the condition,
       let $v = o$
4:     $Q \leftarrow \{\rho \in R' : v_\rho \in V_v\}, \mathcal{Q} \leftarrow \mathcal{Q} \cup \{Q\}, R' \leftarrow R' \setminus Q$.
```

It is easy to see that $\mathcal{Q}$ form a partition of $R'_i$. Focus on any iteration of the loop, and let $v$ be the vertex chosen in the iteration. We argue that we have $\mathsf{mst}'_v(R') + d(o,v) \leq 8F$, implying that the set $Q$ added in the iteration has $\mathsf{mst}(Q) = \mathsf{mst}'_v(R') + d(o,v) \leq 8F$. To see this, assume the two children of $v$ in $T$ are $u$ and $u'$. (If $v$ is a leaf the statement is trivial.) By our choice of $v$ we have $\mathsf{mst}'_u(R') < 3F$ and $\mathsf{mst}'_{u'}(R') < 3F$. Then we have $\mathsf{mst}_v(R') \leq \mathsf{mst}'_u(R') + \mathsf{mst}'_{u'}(R') + d(o,v) + d(v,u) + d(v,u') \leq 3F + 3F + d(o,u) + d(o,u') \leq 6F + F + F = 8F$. The first inequality may not hold with equality since $(v,u)$ or $(v,u')$ may not be in $\mathsf{mst}'_v(R')$.

It remains to prove that for every edge $e \in E$, at most $\left\lceil \frac{\mathsf{mst}'_e(R'_i)}{3F} \right\rceil \leq \left\lceil \frac{\mathsf{mst}_e(R'_i)}{3F} \right\rceil = c_{3F}(R'_i, e)$ groups $Q \in \mathcal{Q}_i$ use the edge $e$. Consider any edge $e$ and assume the group $Q$ is constructed in an iteration in which the vertex we choose is $v$. If $v \in V_e$, then $\mathsf{mst}'_e(R')$ is reduced by at least $3F$ in the iteration since all the edges in $\mathsf{mst}'_v(R')$ are removed from $\mathsf{mst}'_e(R')$. Otherwise, $v$ must be an ancestor of the parent end-vertex of $e$. In this case, $\mathsf{mst}'_e(R')$ becomes $\emptyset$ and thus this is the last time $e$ is used. Moreover once $\mathsf{mst}'_e(R')$ becomes 0, there are no requests in $V_e$. Therefore, $e$ is used at most $\left\lceil \frac{\mathsf{mst}'_e(R'_i)}{3F} \right\rceil$ times. ◀

## 4.4 Scheduling Groups Greedily

Once we partitioned each bundle $R'_i$ into many groups $\mathcal{Q}_i$, we can then treat the groups as jobs and the $k$ vehicles as $k$ machines. Each group $Q \in \mathcal{Q}_i$ can be viewed as a job of size $2\mathsf{mst}(Q)$ that is released at time $(i+2)F$. We need to analyze the maximum flow time achieved using the FIFO algorithm.

Now we define the goal more formally in the language of the scheduling problem. A job $j$ arrives time $r_j$, and upon its arrival, we know its processing time $p_j$. We need to schedule the jobs on the $k$ machines non-preemptively so as to minimize the maximum flow time over all jobs, where the flow time of a job is its completion time minus its release time. We need to design an online algorithm: Once we started processing a job on a machine, we need to complete the job on the machine. Needless to say, the decisions made at or before time $t$ can only depends on the jobs arrived at or before time $t$.

We consider the simple FIFO (first-in-first-out) algorithm: whenever there is an idle machine and an available job (an arrived job that is not being processed), we process the available job with the earliest releasing time on the machine. The following simple lemma is implicit in the analysis of FIFO on scheduling to minimize the maximum flow time:

▶ **Lemma 15.** *Let $P$ be the maximum processing times of all jobs and $D \geq 0$. Assume for any two time points $a \leq b$, the total size of jobs released in time $[a, b]$ is at most $k(b-a)+D$, then the maximum flow time achieved by FIFO is at most $\frac{D}{k} + \frac{2(k-1)P}{k}$.*

Notice that when $k = 1$, the upper bound is simply $D$, and it is a folklore that the FIFO algorithm is optimum.

**Proof.** Focus on a job $j$ and let $s_j$ be its starting time in the schedule produced by the greedy algorithm. Let $t \leq s_j$ be the latest time such that some machine is idle in $(t-\epsilon, t)$ for some positive $\epsilon$. (It is possible that $t = 0$.) Notice that we have $t \leq r_j$ since otherwise $j$ could be started on that machine at time $t - \epsilon$.

All the $k$ machines are busy in time $(t, s_j)$. All but at most $k-1$ jobs that are processed fully or partially in $(t, s_j)$ are released in $[t, r_j]$. The total size of the jobs (including the at most $k-1$ jobs) is at most $(k-1)P + k(r_j - t) + D$, by the conditions of the lemma. Therefore we have $k(s_j - t) + p_j \leq (k-1)P + k(r_j - t) + D$, which is $s_j + \frac{p_j}{k} \leq \frac{(k-1)P}{k} + r_j + \frac{D}{k}$. This implies the flow time of $j$ is $s_j + p_j - r_j \leq \frac{(k-1)P}{k} + \frac{D}{k} + \frac{(k-1)}{k}p_j \leq \frac{D}{k} + \frac{2(k-1)P}{k}$. ◀

We then use Lemma 15 to bound the maximum flow time for the food delivery problem. By Lemma 14, all jobs (which correspond to groups) have size at most $16F$. By Lemma 14 and Theorem 10, the total size of jobs released at any interval $[t, t']$ is at most $(t'-t)k + 4kF$. Therefore, using the greedy algorithm and Lemma 15, every job will have flow time at most $\frac{4kF}{k} + 2 \times 8F = 20F$. Notice that requests in $R'_i$ has arrival time at least $(i-2)F$ and is released at time $(i+2)F$. So, every request has flow time at most $20F + 4F = 24F$. This finishes the proof of Theorem 2 for general $k$, except for the proof of the running time of Algorithm 1, which we argue next.

Assuming all lengths are integers. Then it is easy to design a pseudo-polynomial time algorithm to find a partition $(R_i^{\text{left}}, R_i^{\text{right}})$ of $R_i$ to minimize $\mathsf{cost}(R_i^{\text{left}}|R_{i-1}) + \mathsf{cost}(R_i^{\text{right}}|R_{i+1})$ using dynamic programming. Consider any two edges $e'$ and $e$ such that $e'$ is an ancestor of $e$. Then given $\mathsf{mst}_e(R_i^{\text{left}} \cup R_{i-1}) - \mathsf{mst}_e(R_{i-1})$, $c(R_i^{\text{left}}, e'|R_{i-1})$ does not depend on the set $\left\{\rho \in R_i^{\text{left}} : v_\rho \in V_e\right\}$. Similarly, given $\mathsf{mst}_e(R_i^{\text{right}} \cup R_{i+1}) - \mathsf{mst}_e(R_{i+1})$, $c(R_i^{\text{right}}, e'|R_{i+1})$ does not depend on the set $\left\{\rho \in R_i^{\text{right}} : v_\rho \in V_e\right\}$. Therefore, we can design a dynamic programming where for each $e$ and two integers $M_{\text{left}}$ and $M_{\text{right}}$, we have a cell indicating if there is a partition $(R_i^{\text{left}}, R_i^{\text{right}})$ of $R_i$ such that $\mathsf{mst}_e(R_i^{\text{left}} \cup R_{i-1}) - \mathsf{mst}_e(R_{i-1}) = M_{\text{left}}$ and $\mathsf{mst}_e(R_i^{\text{right}} \cup R_{i+1}) - \mathsf{mst}_e(R_{i+1}) = M_{\text{right}}$. To compute the value of a cell $(e, M_{\text{left}}, M_{\text{right}})$, we look at all cell $(e', M'_{\text{left}}, M'_{\text{right}})$s with $e'$ being a *child* of $e$, and check if there's a way to combine the $M'_{\text{left}}$s (resp. $M'_{\text{right}}$s) to get $M_{\text{left}}$ (resp. $M_{\text{right}}$). (To make this step fast we may assume the input tree is a binary tree: this can be done by adding zero-length dummy edges and vertices) To make the algorithm truly polynomial, we can round all edge lengths to integer multiples of $\epsilon F/(n|R|)$ for any small constant $\epsilon > 0$. Since every one of the $n$ edges is used by at most $|R|$ requests, the total error incurred due to the rounding is at most $\epsilon F/(n|R|) \cdot n|R| = \epsilon F$, which can be ignored.

## References

1 Kemal Altinkemer and Bezalel Gavish. Technical note - heuristics for delivery problems with constant error guarantees. *Transportation Science*, 24:294–297, 1990.

2 Norbert Ascheuer, Sven O Krumke, and Jörg Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 639–650. Springer, 2000.

3 Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'09, pages 1238–1244, USA, 2009. Society for Industrial and Applied Mathematics.

4 Nikhil Bansal, Ho-Leung Chan, Rohit Khandekar, Kirk Pruhs, Baruch Schieber, and Cliff Stein. Non-preemptive min-sum scheduling with resource augmentation. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 614–624. IEEE, 2007.

5 Nikhil Bansal, Moses Charikar, Sanjeev Khanna, and Joseph Naor. Approximating the average response time in broadcast scheduling. In *SODA*, volume 5, pages 215–221. Citeseer, 2005.

6 Nikhil Bansal and Bouke Cloostermans. Minimizing maximum flow-time on related machines. *Theory of Computing*, 12(1):1–14, 2016.

7 Yair Bartal and Shan Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 558–559, 2000.

8 Jatin Batra, Naveen Garg, and Amit Kumar. Constant factor approximation algorithm for weighted flow time on a single machine in pseudo-polynomial time. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 778–789. IEEE Computer Society, 2018.

9 Marcin Bienkowski, Artur Kraska, and Hsiang-Hsuan Liu. Traveling repairperson, unrelated machines, and other stories about average completion times. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2021.

10 Jannis Blauth, Vera Traub, and Jens Vygen. Improving the approximation ratio for capacitated vehicle routing. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 1–14. Springer, 2021.

11 Jessica Chang, Thomas Erlebach, Renars Gailis, and Samir Khuller. Broadcast scheduling: algorithms and complexity. *ACM Transactions on Algorithms (TALG)*, 7(4):1–14, 2011.

12 Moses Charikar and Balaji Raghavachari. The finite capacity dial-a-ride problem. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 458–467. IEEE, 1998.

13 Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Minimizing maximum response time and delay factor in broadcast scheduling. In *European Symposium on Algorithms*, pages 444–455. Springer, 2009.

14 Anamitra Roy Choudhury, Syamantak Das, Naveen Garg, and Amit Kumar. Rejecting jobs to minimize load and maximum flow-time. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1114–1133. SIAM, 2014.

15 Vincent Cohen-Addad, Arnold Filtser, Philip N Klein, and Hung Le. On light spanners, low-treewidth embeddings and efficient traversing in minor-free graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 589–600. IEEE, 2020.

16 Aparna Das and Claire Mathieu. A quasi-polynomial time approximation scheme for euclidean capacitated vehicle routing. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 390–403. SIAM, 2010.

17 Leah Epstein and Rob van Stee. Optimal on-line flow time with resource augmentation. *Discrete applied mathematics*, 154(4):611–621, 2006.

18 Uriel Feige, Janardhan Kulkarni, and Shi Li. A polynomial time constant approximation for minimizing total weighted flow-time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1585–1595. SIAM, 2019.

**19** Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.

**20** Inge Li Gørtz. Hardness of preemptive finite capacity dial-a-ride. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 200–211. Springer, 2006.

**21** Inge Li Gørtz, Viswanath Nagarajan, and R Ravi. Minimum makespan multi-vehicle dial-a-ride. In *European Symposium on Algorithms*, pages 540–552. Springer, 2009.

**22** Xiangyu Guo, Kelin Luo, Zhihao Gavin Tang, and Yuhao Zhang. The online food delivery problem on stars. *Theoretical Computer Science*, 2022. doi:10.1016/j.tcs.2022.06.007.

**23** Anupam Gupta, MohammadTaghi Hajiaghayi, Viswanath Nagarajan, and Ramamoorthi Ravi. Dial a ride from k-forest. *ACM Transactions on Algorithms (TALG)*, 6(2):1–21, 2010.

**24** Mordecai Haimovich and Alexander HG Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of operations Research*, 10(4):527–542, 1985.

**25** Dawsen Hwang and Patrick Jaillet. Online scheduling with multi-state machines. *Networks*, 71(3):209–251, 2018.

**26** Sungjin Im, Shi Li, Benjamin Moseley, and Eric Torng. A dynamic programming framework for non-preemptive scheduling problems on multiple machines. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1070–1086. SIAM, 2014.

**27** Aditya Jayaprakash and Mohammad R Salavatipour. Approximation schemes for capacitated vehicle routing on graphs of bounded treewidth, bounded doubling, or highway dimension. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 877–893. SIAM, 2022.

**28** Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM (JACM)*, 47(4):617–643, 2000.

**29** Bala Kalyanasundaram, Kirk Pruhs, and Mahe Velauthapillai. Scheduling broadcasts in wireless networks. In *European Symposium on Algorithms*, pages 290–301. Springer, 2000.

**30** Anna R Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric tsp. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 32–45, 2021.

**31** Hans Kellerer, Thomas Tautenhahn, and Gerhard J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 418–426. ACM, 1996. doi:10.1145/237814.237989.

**32** Michael Khachay and Roman Dubinin. Ptas for the euclidean capacitated vehicle routing problem in $r^d$. In *International Conference on Discrete Optimization and Operations Research*, pages 193–205. Springer, 2016.

**33** Sven O Krumke, Willem E de Paepe, Diana Poensgen, Maarten Lipmann, Alberto Marchetti-Spaccamela, and Leen Stougie. On minimizing the maximum flow time in the online dial-a-ride problem. In *International Workshop on Approximation and Online Algorithms*, pages 258–269. Springer, 2005.

**34** Sven O Krumke, Willem E De Paepe, Diana Poensgen, and Leen Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295(1-3):279–294, 2003.

**35** Sven Oliver Krumke, Luigi Laura, Maarten Lipmann, Alberto Marchetti-Spaccamela, Willem de Paepe, Diana Poensgen, and Leen Stougie. Non-abusiveness helps: An $o(1)$-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem. In *APPROX*, pages 200–214. Springer, 2002.

**36** Martine Labbé, Gilbert Laporte, and Hélene Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39(4):616–622, 1991.

**37** Giorgio Lucarelli, Benjamin Moseley, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling on unrelated machines with rejections. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, pages 291–300, 2018.

38 Giorgio Lucarelli, Benjamin Moseley, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling to minimize weighted flow-time on unrelated machines. *arXiv preprint*, 2018. `arXiv:1804.08317`.

39 Giorgio Lucarelli, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling in a resource augmentation model based on duality. In *European Symposium on Algorithms (ESA 2016)*, volume 57(63), pages 1–17, 2016.

40 Monaldo Mastrolilli. Scheduling to minimize max flow time: Off-line and on-line algorithms. *International Journal of Foundations of Computer Science*, 15(02):385–401, 2004.

41 Claire Mathieu and Hang Zhou. A ptas for capacitated vehicle routing on trees. *arXiv preprint*, 2021. `arXiv:2111.03735`.

42 Cynthia A Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 140–149, 1997.

43 Lars Rohwedder and Andreas Wiese. A $(2 + \varepsilon)$-approximation algorithm for preemptive weighted flow time on a single machine. *arXiv preprint*, 2020. `arXiv:2011.05676`.

44 Yuanxiao Wu and Xiwen Lu. Capacitated vehicle routing problem on line with unsplittable demands. *Journal of Combinatorial Optimization*, pages 1–11, 2020.