Towards Fairness in Online Service with K Servers and Its Application on Fair Food Delivery

Daman Deep Singh, Amit Kumar, Abhijnan Chakraborty

Department of Computer Science and Engineering,
Indian Institute of Technology Delhi, India
{Daman.Deep.Singh, abhijnan, amitk}@cse.iitd.ac.in

Abstract

The k-SERVER problem is one of the most prominent problems in online algorithms with several variants and extensions. However, simplifying assumptions like instantaneous server movements and zero service time has hitherto limited its applicability to real-world problems. In this paper, we introduce a realistic generalization of k-SERVER without such assumptions – the k-FOOD problem, where requests with source-destination locations and an associated pickup time window arrive in an online fashion, and each has to be served by exactly one of the available k servers. The k-FOOD problem offers the versatility to model a variety of real-world use cases such as food delivery, ride sharing, and quick commerce. Moreover, motivated by the need for fairness in online platforms, we introduce the FAIR k-FOOD problem with the *max-min* objective. We establish that both k-FOOD and FAIR k-FOOD problems are strongly NP-hard and develop an optimal offline algorithm that arises naturally from a time-expanded flow network. Subsequently, we propose an online algorithm DOC4FOOD involving virtual movements of servers to the nearest request location. Experiments on a real-world food-delivery dataset, alongside synthetic datasets, establish the efficacy of the proposed algorithm against state-of-the-art fair food delivery algorithms.

Introduction

The k-SERVER problem (Manasse, McGeoch, and Sleator 1990) is one of the most studied problems in the domain of online algorithms. In this problem, a sequence of requests arrives online at various locations in a m-point metric space and each request has to be served by one of the k servers by moving the server to the corresponding location, and the objective is to minimize the total movement of the servers. Owing to its significance, a number of variants of this problem have been explored in the past. For instance, the k-TAXI problem (Coester and Koutsoupias 2019) extends the k-SERVER problem to consider each request as a pair of points in the metric space. A server must move from a request's source point to the corresponding destination point in order to fulfill the request. The goal is to efficiently assign the k taxis to minimize the total travel distance. Another notable variant is the k-server with Time Windows (k-SERVERTW) problem (Gupta, Kumar, and Panigrahi 2022b) where each request, additionally, has a deadline associated with it within which it needs to be served, allowing a server to handle several 'live' requests in a single visit. Many of these extensions have the capacity to model specific problems like caching, path planning, and resource allocation.

However, all existing variants of the k-SERVER problem assume that the server movement is instantaneous, i.e., once a server is assigned to a particular request, it takes no time for it to move to the request location. Moreover, there is no service time associated with a request, and thus all k servers are available to serve a request at time t even if some of them were assigned a request at time $t-1$. Such simplifying assumptions limit the applicability of k-SERVER problem to more realistic scenarios.

To overcome these issues, in this work, we introduce a general problem, called the k-FOOD problem, which builds upon a number of aforementioned k-server extensions but is more rooted in reality. In this problem, each request corresponds to a pair of points – source and destination – in a metric space accompanied by a pick-up (or preparation) time window. Serving a request involves moving one of the k servers to its source location within the pick-up time window and subsequently moving to its destination. Importantly, the servers take finite amount of time to travel, during which they are unavailable to serve a new request. The objective of k-FOOD is still to minimize the net server movement.

Going further, motivated by the recent reports highlighting the difficult condition of gig delivery drivers in the global south (Gupta et al. 2022; Nair et al. 2022; Sühr et al. 2019; Cao, Wang, and Li 2021; Singh, Das, and Chakraborty 2023), particularly their struggle to earn even minimum wage, we also introduce a variant of k-FOOD problem, called the FAIR k-FOOD problem which assumes a *max-min* objective instead of the *min* cost objective. This objective, inspired by Rawls' theory of justice (Rawls 1971), aims to maximize the minimum reward earned by any server. We demonstrate the applicability of FAIR k-FOOD problem in ensuring fairness in food delivery platforms.

Today, platforms like DoorDash, Deliveroo and Zomato have become de facto destinations for ordering food. Apart from serving many customers, they also provide livelihood to millions of delivery drivers worldwide. Although multiple approaches have been proposed to ensure fair driver assignment (Gupta et al. 2022; Nair et al. 2022), they all adopt a *semi-online* approach where they collect requests within an accumulation time window and then apply an offline algorithm to match with eligible drivers. The underlying problem, however, is inherently online, where food orders (requests) arrive one by one and have to be assigned to one of the eligible drivers (servers). In this work, apart from developing an offline solution scalable up to thousands of requests and hundreds of servers, we propose the first *purely online* driver assignment algorithm for food delivery, which we call DOC4FOOD. Extensive experiments on synthetic and real food-delivery data establish the superiority of DOC4FOOD compared to semi-online solutions.

Our Contributions. In summary, in this paper, we

- introduce the k-FOOD and FAIR k-FOOD problems with the potential to model multiple real-world applications, and show that both problems are strongly NP-hard;
- design a fractional offline-optimal algorithm for the FAIR k-FOOD problem utilizing the corresponding time-expanded flow network;
- propose an online algorithm DOC4FOOD for fair food delivery, employing a prominent heuristic in online algorithms informed by domain-specific knowledge; and
- present extensive experimental analysis on a real-world food delivery dataset and two synthetic datasets.

A detailed version of our work, inclusive of detailed proofs, is available at (Singh, Kumar, and Chakraborty 2023).

Related Works

k-SERVER and its variants. The online k-SERVER problem (Manasse, McGeoch, and Sleator 1990) is arguably the most prominent problem in online algorithms. Over the past few decades, numerous variations of this problem have been explored, including paging (Fiat et al. 1991), k-sever with time windows (Gupta, Kumar, and Panigrahi 2022b), delayed k-server (Bein et al. 2005), k-server with rejection (Bittner, Imreh, and Nagy-György 2014), online k-taxi (Coester and Koutsoupias 2019), Stochastic k-server (Dehghani et al. 2017) and, k-server with preferences (Castenow et al. 2022). However, in all these variants, server movement is always instantaneous. Our proposed k-FOOD problem moves beyond this assumption and captures the subtleties of real-world settings.

Fairness in k-SERVER. The k-SERVER problem and its variants have traditionally been studied with the objective of minimizing the total movement cost. To our knowledge, the only other work that presents a fairness-motivated objective is (Chiplunkar et al. 2023), where the paging problem – a special case of the k-SERVER problem – is studied with the *min-max* objective. They design a deterministic $\mathcal{O}(k \log(n) \log(k))$-competitive algorithm and an $O(\log^2 n \log k)$-competitive randomized algorithm for the online min-max paging problem.

Fairness in online platforms. The growing prevalence of online platforms in various domains, including ride hailing, food delivery and e-commerce, has attracted increasing attention to these research areas (Gupta et al. 2023; Chakraborty et al. 2017; Joshi et al. 2022; Sühr et al. 2019). For instance, research on ride hailing has focused on efficiency maximization (Ta et al. 2017; Jia, Xu, and Liu 2017) and more recently on promoting fairness (Sühr et al. 2019; Cao, Wang, and Li 2021). Online vehicle routing problem (Bertsimas, Jaillet, and Martin 2019) is an interesting work that bears resemblance to our k-FOOD problem. They developed an efficient offline mixed-integer optimization framework that scales well to real-world workload via sparsification and re-optimization of the offline optimal.

Similarly, research on food delivery has seen similar shift from increasing efficiency by minimizing travel costs (Joshi et al. 2022; Yildiz and Savelsbergh 2019; Zeng, Tong, and Chen 2019) to developing equitable food delivery algorithms (Gupta et al. 2022; Nair et al. 2022; Singh, Das, and Chakraborty 2023). Yet, limited exploration exists regarding purely online solutions tailored to fair food delivery. We attempt to fill this gap in the current work.

Problem Statement

Next, we formally describe the classical k-SERVER problem and one of its extensions – k-SERVERTW problem, and then introduce the k-FOOD and FAIR k-FOOD problems.

Definition 1. (The k-SERVER problem) Consider an m-point metric space $(\mathcal{M}, d)$, an online sequence of requests $\sigma = \{r_1, r_2, \cdots, r_n\}$, and a set of k servers existing at specific, not necessarily distinct, points in the metric space. Each request r_i arrives at a specific point in the metric space and must be served by one of the k servers by moving the server to the corresponding location. The movement of a server incurs a cost equivalent to the distance between the current server location and the requested location. The objective is to minimize the total movement cost. The server movement between any two points is assumed to be instantaneous. Hence, whenever a new request r_i arrives, all k servers are immediately available for assignment.

Definition 2. (The k-SERVERTW problem) The k-server with Time Windows (k-SERVERTW) problem extends the k-SERVER problem to accommodate an additional *deadline* associated with a request. Specifically, each request r arriving at a point in the metric space at some time t_r^b with a deadline $t_r^e \geq t_r^b$ must be served by moving one of the k servers to the corresponding location within the time window $[t_r^b, t_r^e]$. The non-triviality (beyond k-SERVER) of this problem lies in the fact that several requests at the same location can be served by a single server visit to this location.

Problem 1. (The k-FOOD problem) Consider a metric space $(\mathcal{M}, d)$ comprising m points. k servers exist at specific points of $\mathcal{M}$, constituting the initial server configuration. Requests from a predefined sequence $\sigma = \{r_1, r_2, \cdots, r_n\}$ arrive one-by-one at specific locations in $\mathcal{M}$. Each request r_j is a 4-tuple (s_j, d_j, t_j^b, t_j^e). Here s_j, d_j are points in $\mathcal{M}$ and $[t_j^b, t_j^e]$ is the time-window associated with the request, also known as the *pick-up* (or *preparation*) time window. Specifically, request r_j arrives at its source s_j at time t_j^b and is considered served if one of the k servers can reach s_j before the deadline t_j^e and subsequently move to the destination d_j. Note that the source-to-destination travel for each request is fixed and not subject to any deadline.

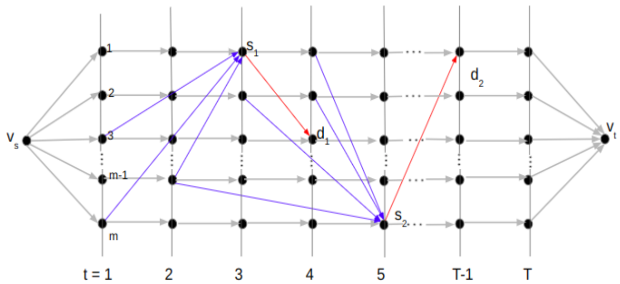

Figure 1: Time-Expanded (or Time-Indexed) Flow Network.

Unlike traditional k-SERVER setting, the k-FOOD problem considers travel time for server movements. When serving a request, the corresponding server becomes temporarily unavailable for other requests and receives a reward equal to the distance it moves. The primary objective of the k-FOOD problem is to minimize the net reward earned (or total distance traveled) by all servers.

Problem 2. (The FAIR k-FOOD problem) The FAIR k-FOOD problem is a variant of the k-FOOD problem with a *maxmin* objective, aiming to maximize the minimum reward earned by any server. The choice of this fairness objective, inspired by the Rawlsian maxmin doctrine (Rawls 1971), is driven by the need to address a real challenge faced by gig delivery drivers (Nair et al. 2022). Recent reports by FairWork (Fairwork 2022, 2023) highlight the hardships faced by drivers across different gig delivery platforms (including food delivery, quick-commerce, e-commerce), revealing that many drivers fail to earn minimum wage mandated by the respective governments. Thus, there is a need to push the minimum income above a certain threshold.[1]

Note that for both k-FOOD problem and FAIR k-FOOD problem, all server movements are on the shortest paths.

Hardness Results

Next, we prove that both k-FOOD and FAIR k-FOOD problems are strongly NP-hard by reductions from the PAGETW and the M-PARTITION (Garey and Johnson 1978) problems respectively, which are known to be strongly NP-hard.

Definition 3. (The PAGETW problem) Given a computing process working on n pages of data, with access to two memory levels: a fast cache capable of holding $k < n$ pages, and a slower memory (e.g., disk) containing all n pages. Initially, all pages reside in the slower memory. Each page p carries a weight w_p. When the process accesses (or requests) a page, it's either fetched from the cache or prompts a page fault, requiring it to enter the cache and potentially evicting an existing page – termed as *serving* a page request. Each page request comes with a deadline by which it must be served. The objective is to minimize the total weight of evicted pages while satisfying the specified deadlines.

Theorem 1. *The k-FOOD problem is NP-hard.*

Proof. The *unit-weight* PAGETW problem, where all pages p have $w_p = 1$, was shown to be NP-hard by (Gupta, Kumar, and Panigrahi 2022a). We can extend this result by presenting a polynomial-time reduction from unit-weight PAGETW problem to a k-FOOD problem instance on uniform metric space with infinite server speeds, thus proving that the k-FOOD problem is NP-hard. A comprehensive proof is available in (Singh, Kumar, and Chakraborty 2023). □

Theorem 2. FAIR *k-FOOD problem is strongly NP-hard.*

Proof. We start by defining a special case of FAIR k-FOOD with $t_r^b = t_r^e$ for all requests r and negligible service time. We call this problem the FAIR k-TAXI problem as it resembles the k-TAXI problem (Coester and Koutsoupias 2019) except the objective. We can show that the FAIR k-TAXI problem is strongly NP-hard using a reduction from the M-PARTITIONproblem. Subsequently, we can extend this result to the FAIR k-FOOD problem. A detailed proof is provided in (Singh, Kumar, and Chakraborty 2023). □

Proposed Methodology

In this section, we describe our offline solution for the FAIR k-FOOD problem, referred to as FLOWMILP. Afterward, we outline various online algorithms for the problem, including our own proposal called DOC4FOOD.

FLOWMILP: Fractional offline optimal for FAIR k-FOOD. We propose a fractional offline solution (FLOWMILP) for the FAIR k-FOOD problem. It is a Mixed Integer Linear Program (MILP) that, intuitively, follows from the min-cost LP formulation for k-SERVER. This MILP may route fractions of servers. Consequently, serving a request entails moving a unit amount of server towards the request's location.

Consider an instance of the FAIR k-FOOD problem with a metric space $(\mathcal{X}, d)$ on m points and a sequence of n requests $\sigma = \{r_1, r_2, \cdots, r_n\}$. We observe the entire duration T, from the arrival of the first order to the end of service of the last order, in timesteps of size η so chosen that each request has a distinct[2] arrival timestep. We now construct a time-expanded graph $\mathcal{G}(V, E)$ where V is the set of nodes obtained by copying the nodes in $\mathcal{X}$ at each timestep i.e., $V = \{v_{i,j} | i \in [m], j \in \{0, \eta, 2\eta, \cdots, T\}\} \cup (v_s, v_t)$ where $v_{i,j}$ is the copy of $v_i \in \mathcal{X}$ at timestep t and E is the set of edges. Here v_s and v_t are special source and sink nodes.

There are three types of edges in E: (i) *Source-Sink edges:* 0-cost edges connecting source node v_s to the nodes in $\{v_{i,0} | i \in [m]\}$ and terminal node v_t to $\{v_{i,T} | i \in [m]\}$; (ii) *Self edges:* 0-cost edges between the nodes $v_{i,j}$ and $v_{i,j+1}$, $\forall i \in [m], j \in \{0, \ldots, (T-1)\}$, indicating that a

[1]Although in this work, we focus on the maxmin objective, the framework and the proposed methodology are versatile enough to accommodate alternate fairness objectives such as *group fairness* (Barocas, Hardt, and Narayanan 2023), where we can aim at minimizing inter-group disparity between servers with distinct sensitive attributes (e.g., gender, race, etc.).

[2]This is without loss of generality. We can arrange simultaneous requests arbitrarily and treat them as distinct arrivals.

server may stay at a location; and (iii) *Cross edges:* For every request $r_j = (s_j, d_j, t_j^b, t_j^e)$, we add the following edges: let $s_{j,t}$ and $d_{j,t}$ denote the copy of s_j and d_j in V at timestep t respectively. First, we add edges from $v_{h,t'} \in V$ to s_{j,t_j^e}; $t_j^b \le t' < t_j^e$, $v_h \neq s_j$ if the time taken to traverse on the shortest path from $v_{h,t'}$ to s_{j,t_j^e} is at most $(t_j^e - t_j^b)$. The cost of this edge is equal to the corresponding shortest path distance. Next, we add an edge between s_{j,t_j^e} and $d_{j,t}$, where $(t - t_j^e)$ is the time taken to travel from s_j to d_j on the shortest path and the cost of this edge is equal to the length of this path. Note that we can prune some edges here. If we have edges from v_{h,t_1} and v_{h,t_2} to s_{j,t_j^e} for some vertex v and times $t_1 < t_2$, then we can remove the first edge. Indeed, there is no loss in generality in assuming that the server arrives at the source location at time t_j^e. This ensures that the server is moved only when the request becomes critical. The flow on each edge e, denoted as f_e, comprises of flow from each of the k servers i.e., $f_e = \Sigma_{i=1}^k f_e^i$. Figure 1 shows an example flow network with 2 requests $r_1 = (s_1, d_1, 1, 3)$ and $r_2 = (s_2, d_2, 2, 5)$.

The FLOWMILP on instance $(\mathcal{G}(V, E), \sigma)$ is defined as

$$\textbf{max. } \mathcal{M} - p.\sum_{r=1}^{n} z_r \tag{1}$$

$$\textbf{s.t. } m_i = \sum_{e \in E} f_e^i.c_e, \forall i \in [k] \tag{2}$$

$$\mathcal{M} \le m_i, \forall i \in [k] \tag{3}$$

$$\sum_{u \in \delta^-(v)} \sum_{i=1}^{k} f_{(u,v)}^i \le 1, \forall v \in \{s_j : j \in [n]\} \tag{4}$$

$$z_{r_j} + \sum_{i=1}^{k} f_{(s_j,d_j)}^i = 1, \forall r_j \in \sigma \tag{5}$$

$$\sum_{v \in \delta^+(u)} f_{(u,v)}^i = \sum_{v \in \delta^-(u)} f_{(u,v)}^i, u \in V, i \in [k] \tag{6}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{k} f_{(s,v_{i1})}^j = \sum_{i=1}^{m} \sum_{j=1}^{k} f_{(v_{iT},t)}^j = k \tag{7}$$

$$\textbf{vars. } f_e^i \in [0,1], \quad \forall i \in [k], \quad z_r \in \{0,1\}, \quad \forall r \in \sigma$$

The objective (1) captures the goal of maximizing the minimum reward $\mathcal{M}$ while minimizing the infeasibilities (z_r's); p being the infeasibility penalty. The binary variable z_r is 1 iff request r cannot be served. The variable f_e^i denotes the flow of server i on edge e. The minimum reward $\mathcal{M}$ is computed using the constraints (2) and (3). Constraints (4) and (5) capture that each request r is served by at most 1 server, and if it is not served, then z_r is 1. Constraints (6) is the flow-conservation constraint, whereas constraint (7) refers to the fact that we have k servers.

Overall, the FLOWMILP formulation has $\mathcal{O}(mk(n+T))$ decision variables and $\mathcal{O}(nkT)$ constraints. The FLOWMILP formulation is flexible and can be easily modified to accommodate various other objectives. For example, the k-FOOD problem can be modeled using FLOWMILP by changing the objective (1) to a min-cost objective and disregarding the separate flows for each server.

Cost Efficiency. Focusing only on maximizing the minimum server reward might result in FLOWMILP deliberately placing servers at locations away from future requests. In real world food delivery setting, such redundant increments in rewards can lead to an unnecessary rise in travel expenses i.e., cost to the platform. This scenario clearly implies an inherent cost-fairness trade-off. To consider both platform and server perspectives, we introduce an extra constraint in the FLOWMILP that upper bounds the total server rewards by a constant multiple of the cumulative edge costs associated with source-destination edges of each request. We choose these edges because they are invariant to the algorithm's other routing decisions.

$$\sum_{i=1}^{k} m_i \le \alpha. \sum_{j=1}^{n} c_{(s_j,d_j)}, \tag{8}$$

The additional constraint is represented as equation (8), where α is a tunable parameter that controls the cost-fairness trade-off. The lower the value of α, the higher the emphasis on reducing the platform-cost and vice-versa. This constraint forces FLOWMILP to be considerate towards both platform and servers. We name this FLOWMILP instance as Two-Sided FLOWMILP (FLOWMILP(2S)).

Online Algorithms. In this work, we consider the following online algorithms pertaining to the *max-min* objective. The guiding principle underlying these approaches is to prioritize an *eligible* server with the minimum accumulated reward while assigning servers to requests. An *eligible* server, with respect to a request r, is one that is *available* (not currently serving any other request) to serve the request as well as *reachable* within r's preparation time.

- **RANDOM.** The core idea here is that as a server starts gaining rewards, it becomes exponentially harder for it to get further rewards. Specifically, an upcoming request r at time t is assigned to one of the eligible servers i with a probability proportional to 2^{-x_i} where x_i is the accumulated reward of the i-th server till time t. It is similar to increasing a server's weight as it accumulates more rewards that, in turn, make its movements harder. The time complexity per request is $\mathcal{O}(k)$.
- **GREEDYMIN.** In this online algorithm, an upcoming request r is assigned to an eligible server with the minimum reward so far. This can be viewed as a specific instance of the RANDOM approach, where the server with the minimum reward is assigned with a unit probability. The time complexity per request is $\mathcal{O}(k)$.
- **DOC4FOOD.** Drawing inspiration from the classical DOUBLE COVERAGE algorithm (Chrobak et al. 1991) commonly seen in the context of k-SERVER problems, which involves proactive movement of idle servers for better performance, we propose **DOuble Coverage for FOOd Delivery (DOC4FOOD)** algorithm that combines GREEDYMIN with a heuristic informed by the domain knowledge specific to the food-delivery sector.

	#Unserved	Cost	Min.R
FLOWMILP	2	2444.50	2444.50
FLOWMILP(2S)	2	1399.56	1399.56
RANDOM	8	1712.04	723
GREEDYMIN	10	1703.43	1065
DOC4FOOD	5	1766.01	1186
MINDELTA	5	1658.12	430
ROUNDROBIN	30	1500.85	10

Table 1: Comparison of FLOWMILP against various online algorithms on the SYNSPARSE dataset.

	#Unserved	Cost	Min.R
FLOWMILP	1	7137.44	7137.44
FLOWMILP(2S)	1	1282.93	1282.93
RANDOM	8	1534.80	800
GREEDYMIN	8	1550.81	1034
DOC4FOOD	7	1554.02	1045
MINDELTA	6	1496.73	855
ROUNDROBIN	12	1494.92	10

Table 2: Comparison of FLOWMILP against various online algorithms on the SYNDENSE dataset.

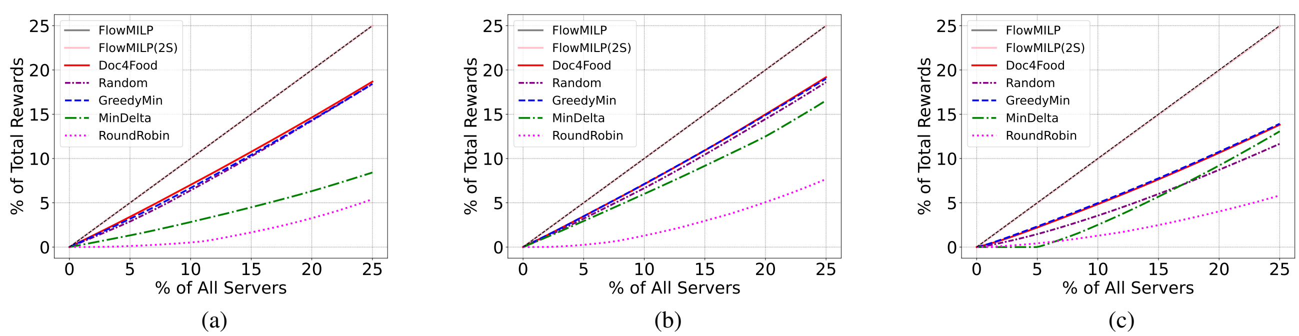

Figure 2: Lorenz curves for various algorithms on the (a) SYNSPARSE, (b) SYNDENSE and (c) food-delivery datasets.

In food delivery, incoming orders originate from a predetermined set of locations corresponding to various restaurants. This set of locations typically forms a small subset of all vertices in the metric space. With respect to k-FOOD, this translates into a distinct subset of points, say $\mathcal{R}$, within the metric space, representing the potential arrival locations for requests.

Transitioning to the DOC4FOOD algorithm, upon the arrival of a request r, akin to the GREEDYMIN strategy, the eligible server with the minimum accumulated reward is chosen to move to r's source location. While it moves towards r, all available servers also move virtually, by a small distance, towards their nearest nodes in $\mathcal{R}$. Such server movements are also referred to as *non-lazy* movements. This mimics the actual practice of delivery drivers' movement to the nearest restaurant location (or market area) when they are idle (Singh, Das, and Chakraborty 2023). Note that while assigning servers, the virtual locations are considered for determining eligibility, but the actual locations are used for calculating rewards for the selected server. For more clarity, the pseudocode for the DOC4FOOD algorithm has been provided in (Singh, Kumar, and Chakraborty 2023).

The consideration of *fractional servers* alongwith the described *edge pruning* allows us to efficiently solve FLOWMILP and FLOWMILP(2S) for many practical instances of the FAIR k-FOOD problem (refer Experimental Evaluation), leveraging advanced MILP solvers (Gurobi Optimization 2023; Cplex 1987). However, in contrast to the offline solutions, the described online algorithms are rounded by default, i.e., they assign an entire server to a single request rather than using fractional assignments.

Experimental Evaluation

In this section, we present the experimental analysis on both synthetic and real-world food-delivery datasets.

Experimental Framework

We conduct experiments on a machine with an Intel(R) Xeon(R) CPU @ 2.30GHz and 252GB RAM running on Ubuntu 20.04.5 LTS. The entire codebase is written in Python 3.9, and the Gurobi optimizer (Gurobi Optimization 2023) is used for solving the linear programs.

Baselines

- FLOWMILP: The fractional offline optimal algorithm for the FAIR k-FOOD problem.
- RANDOM and GREEDYMIN: As explained in the subsection *Online Algorithms* under *Methodology*.
- MINDELTA: A purely online counterpart to the heuristic-based semi-online algorithm developed by (Gupta et al. 2022) aiming to minimize the reward gap between the minimum and maximum earning servers. Takes $\mathcal{O}(k^2)$ time per request assignment.

- ROUNDROBIN: Here, an upcoming request is assigned to the first *eligible* server in round-robin manner. The server assignment complexity per request is $\mathcal{O}(k)$.

Evaluation Metrics

We consider the following evaluation metrics:

- **Number of infeasible requests (#Unserved).** The number of requests that the corresponding algorithm could not serve. This might happen due to a scarcity of *eligible* (available and reachable) servers for the given request.
- **Minimum Reward (Min.R).** Given that our objective is to maximize the minimum reward, we record the minimum reward among the server rewards. The higher the minimum reward, the fairer the algorithm. If it is 0, we look at the number of servers with a 0 reward.
- **Cost.** We define the cost of an algorithm as the average of all the server rewards. Since, for a given request, the source-destination distance is fixed, this cost essentially represents the algorithm's routing and server-assignment decisions. In real-world applications such as food delivery, ride-sharing, etc., it represents the total cost incurred by an online platform to compensate its delivery drivers (also referred to as the *platform-cost*.

Experiments with Synthetic Data

Dataset and Setup. We begin by establishing a graph $\mathcal{X}$ composed of 500 nodes. Using the Erdos-Renyi model (Erdös and Rényi 1959), we add edges with an edge connection probability denoted as p, while ensuring $\mathcal{X}$ remains connected. Subsequently, we generate two distinct datasets: one with $p = 0.5$, referred to as SYNSPARSE, and another with $p = 0.9$, known as SYNDENSE. This deliberate variation enhances our analysis by encompassing different network structures. The edge weights within $\mathcal{X}$ are uniformly selected at random from a set of values $\{10, \ldots, 10000\}$.[3] For each dataset, we generate a total of 250 requests. Recall that each request r in the k-FOOD problem is a 4-tuple (s, d, t^b, t^e). For each request, we sample s, d from $\mathcal{X}$ and t^b, t^e from the interval $[100, 900]$, such that the preparation-time (t^e-t^b) lies in the interval $[1, 100]$. The delivery time of a request is set equal to the distance between the s and d (essentially, the speed of each server is assumed to be 1 unit per timestep). Additionally, we maintain that t^b for each request r is distinct. Consequently, we have a set of 250 requests that arrive at one of the 500 nodes of $\mathcal{X}$ over a span of 1000 timesteps. The choice of the edge weights and the data configuration described above have been inspired by the characteristics of the real-world food-delivery dataset. We consider k=100 servers in each dataset and assume that all the servers are active for the entire duration of 1000 timesteps. We use $\alpha = 1.2$ for the FLOWMILP(2S) algorithm.

Results. Tables 1 and 2 show results on the SYNSPARSE and SYNDENSE datasets respectively. We observe qualitatively similar results for both datasets. The fractional offline algorithms FLOWMILP and FLOWMILP(2S) perform optimally, achieving equal rewards for all servers. As intended, FLOWMILP(2S) reduces the cost while maintaining similar reward distribution compared FLOWMILP. Among online algorithms, DOC4FOOD achieves the highest min. reward while serving a nearly maximal number of requests while MINDELTA, on the other hand, performs poorly in terms of both feasibility and minimum reward maximization. Notably, DOC4FOOD, as intended, reduces infeasibility as compared to GREEDYMIN due to its non-lazy server movements. RANDOM and ROUNDROBIN do well in terms of feasibility but fall short in increasing the minimum reward. Note that a higher (or lower) cost incurred by an algorithm can be primarily due to the more (or lesser) number of requests it serves.

Figures 2(a) and 2(b) depict the Lorenz curves corresponding to various algorithms, focusing on servers within the bottom 25 percentile in terms of rewards. The closer a curve is to the line of equality, the more the fraction of net rewards captured by the corresponding fraction of the servers. The curves for FLOWMILP and FLOWMILP(2S) intersect with the line of equality because all servers earn equal rewards. Remarkably, we see that DOC4FOOD, closely followed by GREEDYMIN, outperforms all other online algorithms, raising the sum of rewards earned by the bottom 25% earners to nearly 20% of the net server rewards.

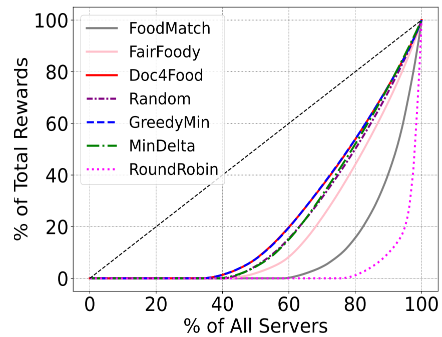

Figure 3: Semi-online algorithms FOODMATCH and FAIRFOODY vs. online algorithms on food delivery dataset.

Experiments with Real Food Delivery Data

Dataset. We utilize a real-world Indian food delivery dataset (Gupta et al. 2022). It comprises 6 days of food-delivery data from 3 major Indian cities, encompassing order load trends for both weekdays and weekends. Notably, the dataset provides thorough information about the delivery drivers (or servers) and orders (or requests) such as delivery vehicle trajectories, road networks of the cities, drivers' chosen working shifts (or active time durations), vehicle IDs, restaurant locations, customer locations, arrival-, pickup-, and delivery-time of each order, among other information.

Additional baselines. In addition to the baseline algorithms described earlier, we consider two offline algorithms from the domain of online food delivery:

[3] Alternate weight selection methods, such as uniform or exponential, yield qualitatively similar results.

	#Unserved	Cost	Min.R.
FLOWMILP	0	19049.20	19049.20
FLOWMILP(2S)	0	8505.19	8505.19
RANDOM	7	6534.46	251
GREEDYMIN	10	6512.85	2490
DOC4FOOD	6	6545.52	2563
MINDELTA	0	6070.41	0(94)
ROUNDROBIN	10	6538.56	11

Table 3: FLOWMILP vs. Online algorithms on the food-delivery dataset. The values in the parentheses indicate the number of servers with 0 reward.

	#Unserved	Cost	Min.R
FOODMATCH	2	8357.47	0(379)
FAIRFOODY	170	9203.52	0(268)
WORK4FOOD	276	8693.66	0(308)
RANDOM	25	15705.28	0(269)
GREEDYMIN	35	8481.59	0(223)
DOC4FOOD	30	8672.00	0(223)
MINDELTA	18	11589.69	0(265)
ROUNDROBIN	56	5431.93	0(263)

Table 4: Offline delivery algorithms vs Online algorithms on the food-delivery dataset. The values in the parentheses indicate the number of servers with 0 reward.

- FOODMATCH: An efficient, heuristic last-mile delivery algorithm introduced by (Joshi et al. 2022).
- FAIRFOODY: A fair food delivery algorithm that tries to achieve an equitable driver income distribution (Gupta et al. 2022).
- WORK4FOOD: A fair food delivery algorithm that attempts to provide minimum wage guarantees to the gig delivery drivers (Nair et al. 2022)

Setup. We present an experimental evaluation on a subset of the dataset comprising the first 8 hours of data of one of the three cities, averaged across all days. It consists of 1050 orders and 734 drivers on the first day. We selected this particular subset because it corresponds to the largest instance of the FLOWMILP problem solvable under the computational constraints of our evaluation framework.[4] The length of each timestep is 1 second. Most orders in the dataset already have distinct arrival times, otherwise we ensure the same by shifting the orders in time up to a few minutes.

The dataset includes information about specific predefined intervals in the day when drivers choose to be active, referred to as "work-shifts". For comparison against FLOWMILP, which assumes the servers to be active for all timesteps, we disregard the work-shifts and consider all the drivers to be active for the entire duration of 8 hours. However, while comparing against more practical algorithms like FOODMATCH and FAIRFOODY, which do consider the work-shifts, we also account for the same. Due to this distinction, we present the comparison between these algorithms and the online algorithms separately.

For the RANDOM baseline, presented evaluations are an average of 5 runs of the algorithm. For the FLOWMILP(2S) algorithm, an α value of 5 is chosen.

Results. Table 3 shows the evaluations of the fractional offline solutions FLOWMILP and FLOWMILP(2S) and the online algorithms. The offline methods achieve optimal solutions where all servers attain equal rewards. FLOWMILP(2S) outperforms FLOWMILP in terms of cost efficiency. Among the online algorithms, DOC4FOOD performs the best in increasing the min. reward, slightly outperforming GREEDYMIN while being more feasible. The RANDOM algorithm shows mediocre performance across all metrics. While MINDELTA excels in minimizing costs, it fares poorly in terms of minimum reward, resulting in nearly 40% servers receiving no rewards.

Table 4 presents the evaluations for the offline algorithms FOODMATCH and FAIRFOODY along with the online baselines. The offline algorithms aren't designed to reject requests beyond preparation time so their corresponding infeasibility values denote pick-up deadline violations; no request was actually infeasible. Again, DOC4FOOD leads to best min. reward slightly outperforming GREEDYMIN that too with fewer infeasbilities. It's worth noting that the online approach MINDELTA demonstrates a close performance to FAIRFOODY, its offline counterpart. Other online algorithms exhibit similar trends as observed in Table 3.

Figure 2(c) and Figure 3 clearly show the superiority of DOC4FOOD in effectively elevating the net rewards of the bottom earners as compared to all other online algorithms.

Conclusion

In this work, we introduce two generalizations of the classical k-SERVER problem – k-FOOD and FAIR k-FOOD– with the ability to model a variety of real-world platforms. We prove these problems to be strongly NP-hard and develop a fractional offline optimal solution FLOWMILP for the FAIR k-FOOD problem. We also propose DOC4FOOD, a heuristic-based pure online algorithm for the food delivery domain. We conduct extensive experimentation on a synthetic dataset as well as a real-world food delivery dataset demonstrating the effectiveness of our proposal.

We hope that our work can serve as a foundation for multiple research problems in both AI and Theoretical Computer Science community. Immediate follow-up works could involve incorporating predictions about future requests to enhance online algorithm performance, particularly in scenarios with a higher volume of requests within a short timespan, and leveraging recent advances in deep learning for constrained optimization to approximate our offline solution.

Reproducibility. Our codebase is available at https://github.com/ddsb01/Fair-kFood.

[4]Scaling to the entire 24-hour duration is feasible through re-optimization techniques (Berbeglia, Cordeau, and Laporte 2010).

References

Barocas, S.; Hardt, M.; and Narayanan, A. 2023. *Fairness and Machine Learning: Limitations and Opportunities*. MIT Press.

Bein, W. W.; Iwama, K.; Larmore, L. L.; and Noga, J. 2005. The delayed k-server problem. In *Fundamentals of Computation Theory: 15th Intl. Symposium*, 281–292.

Berbeglia, G.; Cordeau, J.-F.; and Laporte, G. 2010. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1): 8–15.

Bertsimas, D.; Jaillet, P.; and Martin, S. 2019. Online Vehicle Routing: The Edge of Optimization in Large-Scale Applications. *Operations Research*, 67(1): 143–162.

Bittner, E.; Imreh, C.; and Nagy-György, J. 2014. The online k-server problem with rejection. *Discrete Optimization*, 13.

Cao, Y.; Wang, S.; and Li, J. 2021. The optimization model of ride-sharing route for ride hailing considering both system optimization and user fairness. *Sustainability*, 13(2).

Castenow, J.; Feldkord, B.; Knollmann, T.; Malatyali, M.; and Meyer auf der Heide, F. 2022. The K-Server with Preferences Problem. In *Proc. of 34th ACM Symposium on Parallelism in Algorithms and Architectures*, 345–356.

Chakraborty, A.; Hannak, A.; Biega, A. J.; and Gummadi, K. 2017. Fair sharing for sharing economy platforms. In *Fairness, Accountability and Transparency in Recommender Systems-Workshop on Responsible Recommendation*.

Chiplunkar, A.; Henzinger, M.; Sudhir Kale, S.; and Vötsch, M. 2023. Online Min-Max Paging. In *Proc. of 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1545–1565. SIAM.

Chrobak, M.; Karloof, H.; Payne, T.; and Vishwnathan, S. 1991. New ressults on server problems. *SIAM Journal on Discrete Mathematics*, 4(2): 172–181.

Coester, C.; and Koutsoupias, E. 2019. The Online k-Taxi Problem. In *Proc. of 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, 1136–1147.

Cplex, I. 1987. Ibm ilog cplex optimization studio. *International Business Machines Corporation*, 12(1987-2018).

Dehghani, S.; Ehsani, S.; Hajiaghayi, M.; Liaghat, V.; and Seddighin, S. 2017. Stochastic k-Server: How Should Uber Work? In *44th Intl. Colloquium on Automata, Languages, and Programming (ICALP 2017)*.

Erdös, P.; and Rényi, A. 1959. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6: 290–297.

Fairwork. 2022. Fairwork India Ratings 2022: Labour Standards in the Platform Economy. https://fair.work/en/fw/publications/fairwork-india-ratings-2022-labour-standards-in-the-platform-economy/.

Fairwork. 2023. Fairwork UK Ratings 2023: A Call for Transparency. https://fair.work/en/fw/fairwork-reports/.

Fiat, A.; Karp, R. M.; Luby, M.; McGeoch, L. A.; Sleator, D. D.; and Young, N. 1991. Competitive Paging Algorithms. *J. Algorithms*, 12: 685–699.

Garey, M. R.; and Johnson, D. S. 1978. "Strong" NP-Completeness Results: Motivation, Examples, and Implications. *Journal of ACM*, 25(3): 499–508.

Gupta, A.; Kumar, A.; and Panigrahi, D. 2022a. Caching with Time Windows and Delays. *SIAM Journal on Computing*, 51(4): 975–1017.

Gupta, A.; Kumar, A.; and Panigrahi, D. 2022b. A Hitting Set Relaxation for k-Server and an Extension to Time-Windows. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, 504–515.

Gupta, A.; Nagori, S. J.; Chakraborty, A.; Vaish, R.; Ranu, S.; Nadkarni, P. P.; Dasararaju, N. V.; and Chelliah, M. 2023. Towards Fair Allocation in Social Commerce Platforms. In *Proceedings of the ACM Web Conference 2023*, 3744–3754.

Gupta, A.; Yadav, R.; Nair, A.; Chakraborty, A.; Ranu, S.; and Bagchi, A. 2022. Fairfoody: Bringing in fairness in food delivery. In *Proc. of AAAI Conference on Artificial Intelligence (AAAI)*, volume 36, 11900–11907.

Gurobi Optimization. 2023. Gurobi Optimizer Reference Manual. https://www.gurobi.com.

Jia, Y.; Xu, W.; and Liu, X. 2017. An optimization framework for online ride-sharing markets. In *Proc. of 37th Intl. conference on distributed computing systems*, 826–835.

Joshi, M.; Singh, A.; Ranu, S.; Bagchi, A.; Karia, P.; and Kala, P. 2022. FoodMatch: Batching and Matching for Food Delivery in Dynamic Road Networks. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, 8(1): 1–25.

Manasse, M. S.; McGeoch, L. A.; and Sleator, D. D. 1990. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2): 208–230.

Nair, A.; Yadav, R.; Gupta, A.; Chakraborty, A.; Ranu, S.; and Bagchi, A. 2022. Gigs with Guarantees: Achieving Fair Wage for Food Delivery Workers. In *Proc. of 31st Intl. Joint Conference on Artificial Intelligence*, IJCAI'22.

Rawls, J. 1971. A Theory Of Justice.

Singh, D. D.; Das, S.; and Chakraborty, A. 2023. FairAssign: Stochastically Fair Driver Assignment in Gig Delivery Platforms. In *Proc. of ACM Conference on Fairness, Accountability, and Transparency*, FAccT '23, 753–763.

Singh, D. D.; Kumar, A.; and Chakraborty, A. 2023. Towards Fairness in Online Service with k Servers and its Application on Fair Food Delivery. arXiv:2312.11280.

Sühr, T.; Biega, A. J.; Zehlike, M.; Gummadi, K. P.; and Chakraborty, A. 2019. Two-Sided Fairness for Repeated Matchings in Two-Sided Markets: A Case Study of a Ride-Hailing Platform. In *Proc. of 25th ACM SIGKDD Intl. Conference on Knowledge Discovery & Data Mining*, 3082–3092.

Ta, N.; Li, G.; Zhao, T.; Feng, J.; Ma, H.; and Gong, Z. 2017. An efficient ride-sharing framework for maximizing shared route. *IEEE Transactions on Knowledge and Data Engineering*, 30(2): 219–233.

Yildiz, B.; and Savelsbergh, M. 2019. Provably High-Quality Solutions for the Meal Delivery Routing Problem. *Transportation Science*, 53.

Zeng, Y.; Tong, Y.; and Chen, L. 2019. Last-mile delivery made practical: An efficient route planning framework with theoretical guarantees. *Proc. of VLDB Endowment*, 13(3): 320–333.