Online Matching with Delays and Size-Based Costs

Yasushi Kawase
The University of Tokyo, Japan

Tomohiro Nakayoshi
The University of Tokyo, Japan

Abstract

In this paper, we introduce the problem of *Online Matching with Delays and Size-based Costs (OMDSC)*. The OMDSC problem involves m requests arriving online. At any time, a group can be formed by matching any number of requests that have been received but remain unmatched. The cost associated with each group is determined by the waiting time for each request within the group and size-dependent cost. The size-dependent cost is specified by a penalty function. Our goal is to partition all the incoming requests into multiple groups while minimizing the total associated cost. This problem is an extension of the TCP acknowledgment problem proposed by Dooly et al. (J. ACM, 2001). It generalizes the cost model for sending acknowledgments. This study reveals the competitive ratios for a fundamental case, in which the penalty function takes only values of either 0 or 1. We classify such penalty functions into three distinct cases: (i) a fixed penalty of 1 regardless of the group size, (ii) a penalty of 0 if and only if the group size is a multiple of a specific integer k, and (iii) other situations. The problem in case (i) is equivalent to the TCP acknowledgment problem, for which Dooly et al. proposed a 2-competitive algorithm. For case (ii), we first show that natural algorithms that match all remaining requests are $\Omega(\sqrt{k})$-competitive. We then propose an $O(\log k / \log \log k)$-competitive deterministic algorithm by carefully managing the match size and timing, and prove its optimality. For any penalty function in case (iii), we demonstrate the non-existence of a competitive online algorithm. Additionally, we discuss competitive ratios for other typical penalty functions that are not restricted to take values of 0 or 1.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online matching, competitive analysis, delayed service

Digital Object Identifier 10.4230/LIPIcs.STACS.2025.59

Related Version *Full Version*: `https://arxiv.org/abs/2408.08658` [22]

Funding *Yasushi Kawase*: JST ERATO Grant Number JPMJER2301, JST PRESTO Grant Number JPMJPR2122, JSPS KAKENHI Grant Number JP20K19739, and Value Exchange Engineering, a joint research project between Mercari, Inc. and the RIISE.

Acknowledgements The authors thank anonymous reviewers for useful comments.

1 Introduction

In online gaming platforms, the challenge of matching players for an optimal gaming experience lies in balancing minimal waiting times and high match quality. For instance, consider an online gaming platform hosting a k-player game, such as chess ($k = 2$), Mahjong ($k = 4$), or battle royal-style shooting games (e.g., $k = 60$ for Apex Legends). On such platforms, players enter a waiting queue sequentially, and the platform selects k players from the queue to start a match. To address scenarios with an insufficient number of players, the platform may opt to fill in groups with computer (AI) players. Although this approach ensures prompt matchmaking, it potentially compromises the quality of the gaming experience. Conversely, waiting to gather the full quota of the required k players may significantly increase the waiting time, potentially reducing player satisfaction.

Motivated by the above challenge, we introduce the *Online Matching with Delays and Size-based Costs (OMDSC)* problem. In this problem, requests (corresponding to players) are presented sequentially in real time. At any moment, it is possible to match a group composed of any number of previously received but unmatched requests. The total cost associated with each match is defined by the waiting time for each request within the group and the cost that depends on the group size. The *penalty function* specifies the cost based on the size of the group.

The OMDSC problem has applications beyond gaming. For instance, in online shopping services, product orders arrive sequentially, and the products can be dispatched together at any time. While it is preferable to dispatch as many orders as possible to minimize costs, customers may become frustrated if their wait times are too long. This situation can also be modeled as an OMDSC problem, where the waiting time and the size of each dispatch jointly determine the total cost.

Notably, our problem is closely related to the TCP acknowledgment problem [14], the online weighted cardinality *Joint Replenishment Problem (JRP)* with delays [12], and the online *Min-cost Perfect Matching with Delays (MPMD)* [16]. The TCP acknowledgment problem (without look-ahead) is equivalent to the OMDSC problem with a constant penalty function. Various generalizations of the TCP acknowledgment problem partially capture the OMDSC problem. Specifically, the online weighted cardinality JRP with delays considers the costs dependent on the (weighted) cardinality of each group. In these generalizations, the penalty function is assumed to be monotonically nondecreasing. However, the OMDSC problem treats penalty functions as nonmonotonic. In the MPMD problem, requests arrive on a metric space, and matching is restricted to groups of exactly size 2. The online *Min-cost Perfect k-way Matching with Delays (k-MPMD)* [27] extends the group size to k. For further details, please refer to Section 1.2.

1.1 Our Results

In this study, we determine the competitive ratios of the OMDSC problem for fundamental and critical scenarios in which the penalty function takes only values of either 0 or 1. In the OMDSC problem, dividing a group into multiple smaller groups can reduce penalties. Therefore, it is sufficient to consider a modified penalty function obtained by optimally dividing a group into subgroups. With this modification, we classify penalty functions into three cases: (i) always 1, (ii) 0 if the size is a multiple of k, and (iii) all other scenarios.

In case (i), the OMDSC problem is equivalent to the TCP acknowledgment problem. Dooly et al. [14] proposed a 2-competitive online algorithm that matches all remaining requests when the total waiting cost increases by 1. For case (ii), we first examine natural algorithms that match all the remaining requests whenever a match incurs a positive cost, which we refer to as *match-all-remaining* algorithms. We demonstrate that every match-all-remaining algorithm is $\Omega(\sqrt{k})$-competitive (Theorem 6). Consequently, to obtain an $o(\sqrt{k})$-competitive algorithm, we must consider both the timing and size of the requests to be matched. By carefully managing remaining requests, we propose an $O(\log k/\log\log k)$-competitive online algorithm (Theorem 8). We also prove that this competitive ratio is the best possible (Theorem 15). It is worth mentioning that the competitive ratio for case (ii) with $k = 1$ is trivial (Theorem 3) and that with $k = 2$ can be obtained from the results of Emek et al. [17]. For any penalty function in case (iii), we prove that no competitive online algorithm exists (Theorem 2). The results are summarized in Table 1. Furthermore, the competitive ratios for other typical penalty functions that are not restricted to take values of 0 or 1 are discussed in Section 2.3.

Table 1 Competitive ratios for the OMDSC problem.

Penalty function (with modification)		**Competitive ratio**	**Reference**
(i) always 1		2	Dooly et al. [14]
(ii) 0 if the size is a multiple of k	$k = 1$	1	Theorem 3
	$k = 2$	3	Emek et al. [17]
	general k	$\boldsymbol{\Theta(\log k / \log\log k)}$	Theorems 8 and 15
(iii) other scenarios		**unbounded**	Theorem 2

1.2 Related Work

An online version of the matching problem was first introduced by Karp et al. [21]. In their study, arriving requests are matched immediately upon arrival, primarily focusing on bipartite matching. Additionally, research has been conducted to minimize matching costs, both in settings where vertices have determined costs and in settings that consider distances in a metric space. Mehta et al. [26] proposed an application to AdWords. Other applications, such as food delivery, have also been considered. For further details, please refer to [15, 25].

In applications such as online game matchmaking and ride-sharing services, service providers can delay user matches. Emek et al. [16] modeled this scenario as an MPMD problem. In this setting, the requests can be retained by incurring waiting costs. Several online algorithms have been proposed for solving the MPMD problem [1, 3, 8]. Subsequently, Melnyk et al. [27] extended the MPMD problem to the k-MPMD problem, which requires exactly matching the size k. To address this problem, deterministic and randomized algorithms have been proposed [19, 27].

Dooly et al. [14] introduced the TCP acknowledgment problem that involves acknowledging TCP packets. In this problem, the aim is to minimize the sum of the acknowledgment costs and the costs of delaying TCP packets. Optimal deterministic and randomized algorithms were proposed by Dooly et al. [14] and Karlin et al. [20], respectively. The TCP acknowledgment problem is equivalent to the OMDSC problem, where the penalty function falls under case (i). Conversely, the OMDSC problem can be viewed as a generalization of the acknowledgment cost of the TCP acknowledgment problem.

Various extensions of the TCP acknowledgment problem have been proposed [2, 5–7, 9–12, 29]. In particular, Chen et al. [12] introduced the problem of an online weighted cardinality JRP with delays. Their model can handle penalties based on the size of the match. However, unlike in our study, their penalty function is restricted to monotonically nondecreasing. They proposed a constant-competitive algorithm to solve this problem.

The objectives of the MPMD and k-MPMD problems are to pair requests and form groups of exact size k, respectively. In contrast, the proposed OMDSC problem does not impose constraints on the number of requests in each group. Emek et al. [16] introduced a problem called MPMDfp, which allows the clearance of a request at a cost. The MPMDfp problem on a single source is equivalent to the OMDSC problem where the penalty function corresponds to case (ii) with $k = 2$. Additionally, the MPMDfp problem on a single source can be reduced to an MPMD problem using two sources [16]. For MPMD on two sources, Emek et al. [17] proposed a deterministic algorithm, and He et al. [18] proposed a randomized algorithm (see Section 4 for more details).

Another approach to extend the MPMD problem is to modify waiting costs. Liu et al. [23] generalized the waiting costs from linear to convex. Other variations of waiting costs have also been studied [4, 13, 24]. Pavone et al. [28] investigated *Online Hypergraph Matching with*

Deadlines problem. Although hypergraph matching does not impose constraints on group size, it differs from the OMDSC problem in that it employs deadlines instead of delays, and requests arrive at one at each unit of time.

2 Preliminaries

We denote the set of real numbers, nonnegative real numbers, integers, nonnegative integers, and positive integers by $\mathbb{R}$, $\mathbb{R}_+$, $\mathbb{Z}$, $\mathbb{Z}_+$, and $\mathbb{Z}_{++}$, respectively. In addition, for a positive integer $k \in \mathbb{Z}_{++}$, we define $\mathbb{Z}_k$ as the set of integers from 0 to $k-1$.

2.1 Problem Settings

In this section, we formally define the OMDSC problem. An instance of the problem is specified by a penalty function $f\colon \mathbb{Z}_{++} \to \mathbb{R}_+$ and m requests $V = \{v_1, v_2, \ldots, v_m\}$ that arrive in real time. Each request v_i arrives at time t_i, where $0 \le t_1 \le t_2 \le \cdots \le t_m$ is assumed.

An online algorithm initially knows only the function f; it does not have prior knowledge of the arrival times or the total number of requests. Each request v_i is revealed at time t_i. At each time τ, the algorithm can match any subset S_j of requests that appeared by time τ and have not yet been matched. The cost of matching requests in S_j at time τ_j is defined as

$$f(|S_j|) + \textstyle\sum_{i\colon v_i \in S_j} (\tau_j - t_i)\,,$$

where the first term represents the *size cost* and the second term represents the *waiting cost*. In addition, the *waiting cost* at time τ is defined as $\sum_{i\colon v_i \in V'}(\tau - t_i)$, where V' is the set of unmatched but previously presented requests at that time.

The objective is to design an online algorithm that matches all requests while minimizing the total cost. We use the *competitive ratio* to evaluate the performance of the online algorithm. For a given instance σ, let $\mathrm{ALG}(\sigma)$ represent the cost incurred by the online algorithm and let $\mathrm{OPT}(\sigma)$ denote the optimal cost with prior knowledge of all requests in the instance. The competitive ratio of ALG for an instance σ is defined as $\mathrm{ALG}(\sigma)/\mathrm{OPT}(\sigma)$, interpreting $0/0$ as 1. In addition, the competitive ratio of ALG for a problem is defined as the supremum of the competitive ratios over all instances, that is, $\sup_\sigma \mathrm{ALG}(\sigma)/\mathrm{OPT}(\sigma)$. We call an online algorithm ρ-competitive if the competitive ratio of that algorithm is ρ.

The competitive ratio defined above is the *strict* competitive ratio. We can also define the *asymptotic* competitive ratio as $\limsup_{\mathrm{ALG}(\sigma)\to\infty} \mathrm{ALG}(\sigma)/\mathrm{OPT}(\sigma)$. However, in our problem, the strict and asymptotic competitive ratios of the optimal algorithm coincide. Indeed, if an algorithm is strictly ρ-competitive, then it is also asymptotically at most ρ-competitive by definition. Moreover, if no strictly ρ-competitive algorithm exists, we can construct an instance for each algorithm in which the strict competitive ratio is at least ρ. Thus, by repeatedly constructing and providing such instances, we can conclude that no algorithm has an asymptotic competitive ratio better than ρ. Hence, we only consider the strict competitive ratio hereafter.

2.2 Binary Penalty Function

This study focuses primarily on penalty functions that take values only of either 0 or 1. We discuss other penalty functions in Section 2.3. For such a binary penalty function f, we may be able to match n requests with size cost 0, even if $f(n) = 1$, by appropriately partitioning the requests. For instance, if $f(2) = f(3) = 0$ and $f(7) = 1$, we can match 7 requests with a size cost of 0 by partitioning them into groups of sizes 2, 2, and 3. We introduce a notation for the set of numbers of requests that can be matched with a size cost of 0 as follows:

▶ **Definition 1.** *For the penalty function $f\colon \mathbb{Z}_{++} \to \{0,1\}$, we define the zero-penalty set B_f as a set of quantities that can be matched with a size cost of 0 by optimally dividing requests into subsets and matching them. Formally,*

$$B_f := \left\{ n \in \mathbb{Z}_{++} \;\middle|\; \exists s \in \mathbb{Z}_{++},\, \exists n_1, \dots, n_s \in \mathbb{Z}_{++} \text{ s.t.} \sum_{i=1}^{s} n_i = n,\; f(n_1) = \cdots = f(n_s) = 0 \right\}.$$

For example, if $f(1) = 0$, then $B_f = \mathbb{Z}_{++}$. Alternatively, if $f(1) = 1$ and $f(2) = f(3) = 0$, then $B_f = \mathbb{Z}_{++} \setminus \{1\}$. We can interpret the size cost of matching n requests as 0 if $n \in B_f$ and 1 if $n \notin B_f$.

We classify binary penalty functions into the following three types: (i) $B_f = \emptyset$, (ii) $B_f = \{kn \mid n \in \mathbb{Z}_{++}\}$ for some $k \in \mathbb{Z}_{++}$, and (iii) all other scenarios. Case (i) is the situation in which the penalty for matching requests is always 1 (i.e., $f(n) = 1$ for all $n \in \mathbb{Z}_{++}$). Case (ii) is a situation in which a set of requests can be matched without a size cost only if the size is a multiple of $k = \min\{n \in \mathbb{Z}_{++} \mid f(n) = 0\}$ (i.e., $f(k) = 0$ and $f(n) = 1$ for all $n \in \mathbb{Z}_{++} \setminus \{kn' \mid n' \in \mathbb{Z}_{++}\}$). This case is applicable in the context of an online gaming platform that hosts a k-players game, as described in Introduction.

2.3 Other Penalty Functions

If the range of the penalty function is $\{0, \mu\}$ with a positive real μ, it can be treated as a binary penalty function by scaling the increase rates of the waiting costs μ times slower. This is because the cost of matching requests in S_j at time τ_j can be expressed as $\mu \cdot \big(f(|S_j|)/\mu + \sum_{i:\, v_i \in S_j} (\tau_j/\mu - t_i/\mu)\big)$. For example, if $\mu = 60$ and the unit time is one minute in the original problem, the range of the penalty function can be treated as $\{0,1\}$ by adjusting the unit time to one hour. Let $\mu, \lambda \in \mathbb{R}$ with $0 < \mu \leq \lambda$ and consider a penalty function that takes values of 0 or within the range $[\mu, \lambda]$. By applying our ρ-competitive algorithm to the problem while treating positive penalties as if they were μ, we can obtain a $(\rho \cdot \lambda/\mu)$-competitive algorithm. In addition, our impossibility result for case (iii) can be transferred in the same manner.

Another interesting penalty function is $f(n) = \lceil n/k \rceil$ for a specific integer k. This penalty function appears when a service can process up to k requests simultaneously, and each processing costs a fixed amount. For this penalty function, an algorithm similar to that in (i) is 2-competitive for any $k \geq 2$. We also prove that this competitive ratio is best possible for every $k \geq 2$ (see the full version [22]). For $k = 1$, the algorithm that matches each request upon its arrival is 1-competitive.

3 Zero-penalty Set is Nonempty and Non-representable as Multiples

In this section, we examine case (iii), where the penalty function f satisfies $B_f \neq \emptyset$ and $B_f \neq \{kn \mid n \in \mathbb{Z}_{++}\}$ for any $k \in \mathbb{Z}_{++}$. We demonstrate that, in this case, no algorithm has a finite competitive ratio.

To illustrate this intuitively, let us consider the case where $f(1) = 1$ and $f(2) = f(3) = 0$, representing a poker table for at least two players. Imagine two players arriving initially, followed potentially by a third. If we match the first two players immediately upon their arrival, matching the subsequent player incurs a cost. Alternatively, if we wait to match the first two, an unnecessary waiting cost is incurred when no additional player appears. We can construct similar instances for every penalty function in case (iii).

▶ **Theorem 2.** *For the OMDSC problem with a penalty function f that satisfies both $B_f \neq \emptyset$ and $B_f \neq \{kn \mid n \in \mathbb{Z}_{++}\}$ for every $k \in \mathbb{Z}_{++}$, the competitive ratio of any randomized algorithm is unbounded against an oblivious adversary.*

Proof. Let $k^* := \min B_f$, where such a value must exist by the assumption that $B_f \neq \emptyset$. Additionally, let $\ell := \min(B_f \setminus \{k^* n \mid n \in \mathbb{Z}_{++}\})$, where such a value must exists because $\{k^* n \mid n \in \mathbb{Z}_{++}\} \subseteq B_f$ and $B_f \neq \{k^* n \mid n \in \mathbb{Z}_{++}\}$. We fix an arbitrary online algorithm ALG and take a sufficiently small $\varepsilon > 0$. Consider an instance where k^* requests are given at time 0, and afterward, there may be arrivals of $\ell - k^*$ additional requests at time ε depending on the behavior of ALG. Suppose that ALG matches all the initial requests before ε with probability p.

If $p \geq 1/2$, consider an instance where $\ell - k^*$ additional requests are given at time ε. Since $\ell - k^* \notin B_f$, ALG incurs an expected size cost of at least p ($\geq 1/2$). In contrast, the minimum total cost is $k^*\varepsilon$ by matching all $k^* + (\ell - k^*) = \ell$ requests at time ε. Thus, the competitive ratio is at least $p/(k^*\varepsilon) \geq 1/(2k^*\varepsilon)$.

Conversely, if $p < 1/2$, consider the instance where no additional requests are presented. Then, the (expected) waiting cost for ALG is at least $(1-p)k^*\varepsilon > k^*\varepsilon/2$, while the offline optimal cost is 0 by matching all requests at time 0. Hence, the competitive ratio is unbounded.

Therefore, in both scenarios, the competitive ratio is unbounded as ε goes to 0, proving that the competitive ratio for any online algorithm is unbounded. ◀

4 Zero-penalty Set is Multiples of an Integer

In this section, we investigate the case (ii), where the penalty function f satisfies $B_f = \{kn \mid n \in \mathbb{Z}_{++}\}$ for some $k \in \mathbb{Z}_{++}$.

When $k = 1$ (i.e., $B_f = \mathbb{Z}_{++}$), we have $f(1) = 0$. Thus, the algorithm that immediately matches each request individually upon its arrival incurs a cost of 0 and is 1-competitive.

▶ **Theorem 3.** *For the OMDSC problem with a penalty function f such that $f(1) = 0$, there exists a 1-competitive deterministic online algorithm.*

For the case $k = 2$, the OMDSC problem is equivalent to the problem of MPMDfp in a metric space consisting of a single point. It is known that the MPMDfp problem can be reduced to the MPMD problem by doubling the number of points on the metric space [16]. By this reduction, the OMDSC problem with $k = 2$ can be reduced to the problem of MPMD on two sources by setting the distance between two sources to 2 and giving requests for two sources simultaneously when a request is given in the OMDSC problem. A matching across two sources in the MPMD problem corresponds to a matching of a single request in the OMDSC problem. The optimal cost of the reduced MPMD problem is twice that of the original OMDSC problem.

Emek et al. [17] provided a 3-competitive online algorithm for the online MPMD problem on two sources and demonstrated that this is best possible. In their instances constructed for the proof of the lower bound, requests are always given to two sources simultaneously, which can be reduced to the OMDSC problem with $k = 2$. Therefore, we obtain the following theorem.

▶ **Theorem 4** (Emek et al. [17])**.** *For the OMDSC problem with a penalty function f such that $B_f = \{2n \mid n \in \mathbb{Z}_{++}\}$, there exists a 3-competitive deterministic algorithm. Moreover, no deterministic online algorithm has a competitive ratio smaller than 3.*

He et al. [18] proposed a 2-competitive randomized algorithm against an oblivious adversary for the MPMD problem on two sources. Thus, this leads to a 2-competitive randomized algorithm for the OMDSC problem with $B_f = \{2n \mid n \in \mathbb{Z}_{++}\}$.

In the remainder of this section, we conduct an asymptotic analysis with respect to k.

Firstly, we define a type of algorithm as below and demonstrate the difficulty of case (ii) compared to case (i).

▶ **Definition 5.** *We call an algorithm for the OMDSC problem* match-all-remaining *if, whenever it makes a match, it matches all remaining requests or the size is in B_f.*

For case (i), where a penalty to match requests is always 1, a match-all-remaining algorithm achieves the optimal competitive ratio. However, for case (ii), we show that every match-all-remaining algorithm has a competitive ratio of $\Omega(\sqrt{k})$, where the proof can be found in the full version [22].

▶ **Theorem 6.** *For the OMDSC problem with a penalty function f that satisfies $B_f = \{kn \mid n \in \mathbb{Z}_{++}\}$, every match-all-remaining algorithm has a competitive ratio of $\Omega(\sqrt{k})$.*

In Section 4.1, we provide an $O(\log k/\log\log k)$-competitive algorithm by utilizing partial matchings of remaining requests. Thus, no match-all-remaining algorithm is optimal. Subsequently, in Section 4.2, we establish the lower bound of $\Omega(\log k/\log\log k)$.

Before proceeding, we introduce some notations. Recall that $\mathbb{Z}_k = \{0, 1, \dots, k-1\}$.

▶ **Definition 7.** *We write $\overline{x} \in \mathbb{Z}_k$ to represent the remainder of $x \in \mathbb{Z}$ when divided by k. Additionally, for $\ell, r \in \mathbb{Z}_+$, we define the cyclic interval*

$$[\![\ell, r]\!]_k := \begin{cases} \{x \in \mathbb{Z}_k \mid \overline{\ell} \le x \le \overline{r}\} = \{\overline{\ell}, \overline{\ell}+1, \dots, \overline{r}\} & (\text{if } \overline{\ell} \le \overline{r}), \\ \{x \in \mathbb{Z}_k \mid x \le \overline{r} \text{ or } \overline{\ell} \le x\} = \{\overline{\ell}, \overline{\ell}+1, \dots, k-1, 0, 1, \dots, \overline{r}\} & (\text{if } \overline{\ell} > \overline{r}). \end{cases}$$

With this notation, $a \equiv b \pmod k$ can be expressed as $\overline{a} = \overline{b}$. Note that we have $\left|[\![\ell, r]\!]_k\right| = \overline{r} - \overline{\ell} + 1$ if $\overline{\ell} \le \overline{r}$ and $\left|[\![\ell, r]\!]_k\right| = k + \overline{r} - \overline{\ell} + 1$ if $\overline{\ell} > \overline{r}$.

Let α be a real number such that $\alpha^\alpha = k$. Then, we have $\alpha = \Theta(\log k/\log\log k)$ because the ratio $\frac{\alpha}{\log k/\log\log k} = \frac{\alpha}{\log \alpha^\alpha/\log\log \alpha^\alpha} = \frac{\log\alpha + \log\log\alpha}{\log\alpha}$ approaches 1 as $k \to \infty$ (i.e., $\alpha \to \infty$). Thus, our task is to provide upper and lower bounds of $O(\alpha)$ and $\Omega(\alpha)$, respectively.

4.1 Upper Bound

The objective of this subsection is to prove the following theorem.

▶ **Theorem 8.** *For the OMDSC problem with a penalty function f that satisfies $B_f = \{kn \mid n \in \mathbb{Z}_{++}\}$, there exists an $O(\log k/\log\log k)$-competitive deterministic online algorithm.*

According to Theorem 6, an $O(\alpha)$ $(= O(\log k/\log\log k))$-competitive algorithm must consider both the timing and the size of requests to be matched. To address this requirement, we propose an algorithm that references the costs of other algorithms. The execution of the algorithm is divided into phases. In each phase, the goal is to ensure that any competing algorithm incurs a cost of at least 1, while our algorithm's cost remains at most $O(\alpha)$. We categorize competing algorithms based on the number of unmatched requests they carry over from the previous phase. As for the current phase, we assume that algorithms only perform matches of sizes that are multiples of k, since otherwise, their cost immediately reaches at least 1. Furthermore, it is sufficient to focus on "greedy" algorithms that immediately match k requests whenever at least k unmatched requests exist.

The first phase starts at the beginning. In each phase, our algorithm runs with the following $k+2$ variables.

▶ **Definition 9.** *At each time t within each phase, the variables $W_0(t), W_1(t), \dots, W_{k-1}(t)$, $s(t)$, and $a(t)$ are defined as follows:*

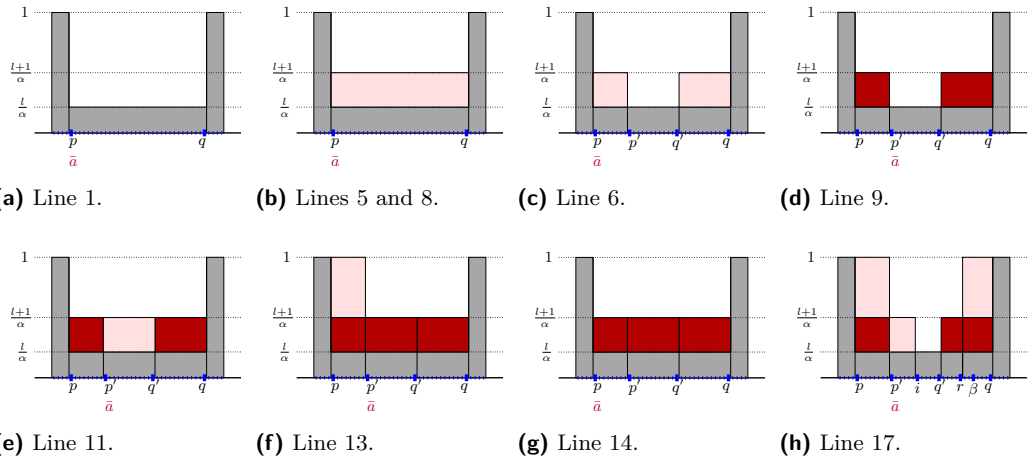
- *$W_i(t)$ $(i \in \mathbb{Z}_k)$: the waiting cost incurred by time t within the phase on the algorithm that greedily performs matches with no size cost, assuming $(\overline{k-i})$ unmatched requests are carried over to the current phase.*
- *$s(t)$: the number of requests given by time t in the phase.*
- *$a(t)$: the number of requests that our algorithm has matched up to, but not including, time t during the phase.*

We will omit the argument t when there is no confusion.

Note that, if $(k-i)$ unmatched requests are carried over to the current phase, the optimum cost incurred thus far within the phase is at least $\min\{1, W_i\}$. At the end of the phase, our algorithm ensures $W_i \geq 1$ for all $i \in \mathbb{Z}_k$. This means that any algorithm incurs a cost of at least 1 per phase because matching a number of requests that is not a multiple of k results in a cost of 1.

During each phase, our algorithm recursively processes a subroutine **recurring**$(\llbracket p, q \rrbracket_k, l)$, which takes as parameters a cyclic interval $\llbracket p, q \rrbracket_k$ and an integer l. When the subroutine is called, it ensures that $W_i \geq 1$ for all $i \notin \llbracket p, q \rrbracket_k$ and $W_i \geq l/\alpha$ for all $i \in \llbracket p, q \rrbracket_k$. In each iteration of the subroutine, the interval size $\left|\llbracket p, q \rrbracket_k\right|$ is reduced by a factor of $\Theta(1/\alpha)$ or l is increased by one, with incurring cost at most $O(1)$. As a result, all W_i reach at least 1 after $O(\alpha)$ recursions by $\alpha^\alpha = k$.

At the beginning of each phase, the variables $W_0, W_1, \dots, W_{k-1}$, s, and a are 0 and **recurring**$(\llbracket 0, k-1 \rrbracket_k, 0)$ is called. Throughout the phase, the algorithm updates the values of $W_0, W_1, \dots, W_{k-1}$, s, and a, appropriately. Moreover, if at least k unmatched requests exist (i.e., $s - a \geq k$), it immediately matches k of them.



(a) Line 1. **(b)** Lines 5 and 8. **(c)** Line 6. **(d)** Line 9.

(e) Line 11. **(f)** Line 13. **(g)** Line 14. **(h)** Line 17.

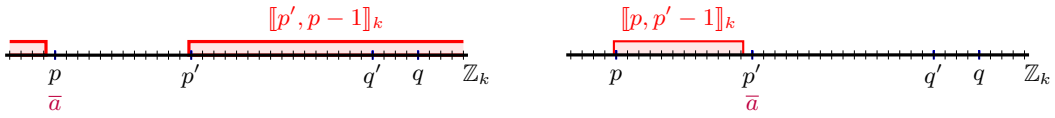
Figure 1 Lower bounds of W_i at each line of Algorithm 1. Gray areas indicate lower bounds of W_i at the beginning of **recurring**$(\llbracket p, q \rrbracket_k, l)$. Red areas illustrate increments earlier than the last wait in **recurring**$(\llbracket p, q \rrbracket_k, l)$. Pink areas represent increments during the last wait.

We now describe the subroutine **recurring**$(\llbracket p, q \rrbracket_k, l)$. See Algorithm 1 for a formal description. Figure 1 illustrates lower bound of each W_i at each line.

When **recurring**$(\llbracket p, q \rrbracket_k, l)$ is called, it is guaranteed that $p, q \in \mathbb{Z}_k$, $l \in \{0, 1, \dots, \lceil \alpha \rceil\}$, $W_i \geq 1$ for all $i \notin \llbracket p, q \rrbracket_k$, $W_i \geq l/\alpha$ for all $i \in \llbracket p, q \rrbracket_k$, and $\overline{a} = p$ (recall that $\overline{a} \in \mathbb{Z}_k$ is the remainder of a when divided by k, see also Figure 1a). The recursion ends (line 3) when $\left|\llbracket p, q \rrbracket_k\right|$ equals to 1 or when l is at least α. Then, it matches all remaining requests and proceeds to the next phase.

Algorithm 1 The pseudo-code of **recurring**$([\![p,q]\!]_k, l)$.

```
/* W_0, W_1, ..., W_{k-1}, s, a are variables that change as defined in Definition 9 */
1  def recurring([[p,q]]_k, l):
       /* Ensure: p,q ∈ Z_k, l ∈ Z_+, W_i ≥ 1 (∀i ∉ [[p,q]]_k), W_i ≥ l/α (∀i ∈ [[p,q]]_k), ā = p */
2      Wait until W_ā (= W_p) increases by 2;                          // Cost: 2
3      if |[[p,q]]_k| = 1 or l ≥ α then
4          Match all remaining requests and Proceed to the next phase; // Cost: ≤ 1
5      if W_i ≥ (l+1)/α for all i ∈ [[p,q]]_k then call recurring([[p,q]]_k, l+1);
6      Let [[p',q']]_k ⊆ [[p,q]]_k be the smallest cyclic interval where W_i ≥ (l+1)/α for all
         i ∉ [[p',q']]_k;
7      Wait until s̄ ∈ [[p', p-1]]_k or W_ā (= W_p) increases by 1/α;   // Cost: ≤ 1
8      if W_ā (= W_p) increased by 1/α at line 7 then call recurring([[p,q]]_k, l+1);
9      Match p'−p requests;                                            // Cost: 1
10     Wait until W_ā (= W_{p'}) increases by 2;                       // Cost: 2
11     if W_i ≥ (l+1)/α for all i ∈ [[p',q']]_k then
12         Wait until s̄ ∈ [[p, p'-1]]_k or W_ā (= W_{p'}) increases by 1; // Cost: ≤ 1
13         if W_ā (= W_{p'}) increased by 1 at line 12 then call recurring([[p',q]]_k, l+1);
14         Match p−p' requests;                                        // Cost: ≤ 1
15         Call recurring([[p,q]]_k, l+1);
16     else
17         Let [[p',r]]_k ⊆ [[p',q]]_k be the smallest cyclic interval where W_i ≥ 1 for all
             i ∉ [[p',r]]_k;
18         Call recurring([[p',r]]_k, l);
```



(a) Situation at line 7.

(b) Situation at line 12.

Figure 2 Relative positions of p, q, p', q', and $\overline{a}$ in Algorithm 1.

At the beginning of each recursion, the algorithm waits until $W_{\overline{a}}$ increases by 2 (line 2). Afterward, if $W_i \geq (l+1)/\alpha$ for all $i \in [\![p,q]\!]_k$, it calls **recurring**$([\![p,q]\!]_k, l+1)$ (Figure 1b). Now, assume that $W_i < (l+1)/\alpha$ for some $i \in [\![p,q]\!]_k$. In this case, there exists a smaller interval $[\![p',q']\!]_k$ ($\subseteq [\![p,q]\!]_k$) such that $W_i \geq (l+1)/\alpha$ for all $i \in [\![p,q]\!]_k \setminus [\![p',q']\!]_k$ (Figure 1c). The algorithm picks the smallest such cyclic interval $[\![p',q']\!]_k$ (line 6).

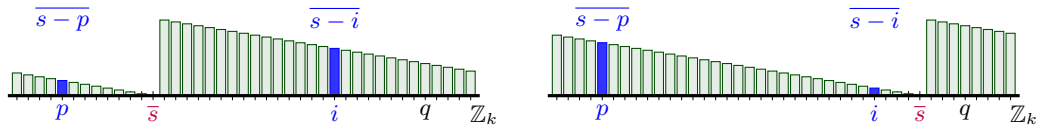
Next, it attempts to change $\overline{a}$ from p to p' by matching $\overline{p'-p}$ requests. To achieve this, it waits to satisfy $\overline{s} \in [\![p', p-1]\!]_k$ (see Figure 2a) until $W_{\overline{a}}$ ($= W_p$) increases by $1/\alpha$ (line 7). If $W_{\overline{a}}$ increases by $1/\alpha$, then $W_i \geq (l+1)/\alpha$ holds for every $i \in [\![p,q]\!]_k$ as we will show later (Figure 1b). Consequently, the algorithm calls **recurring**$([\![p,q]\!]_k, l+1)$ (line 8). Now, suppose that $\overline{s}$ in $[\![p', p-1]\!]_k$ at some point. In that case, it changes $\overline{a}$ from p to p' by matching $\overline{p'-p}$ requests (line 9 and see Figure 1d).

Then, the algorithm waits until $W_{\overline{a}}$ ($= W_{p'}$) increases by 2 (line 10). We consider two cases: (♠) $W_i \geq (l+1)/\alpha$ for all $i \in [\![p',q']\!]_k$ (Figure 1e) and (♡) $W_i < (l+1)/\alpha$ for some $i \in [\![p',q']\!]_k$ (Figure 1h). In case (♠), the algorithm attempts to move $\overline{a}$ from p' to p to call **recurring**$([\![p,q]\!]_k, l+1)$. To do this, the algorithm waits until $\overline{s} \in [\![p, p'-1]\!]_k$ (see Figure 2b)

or until $W_{\overline{a}}$ ($= W_{p'}$) increases by 1 (line 12). If $W_{p'}$ increases by 1, then $W_i \geq 1$ holds for every $i \in [\![p, p']\!]_k$ as we will show later (Figure 1f). Then, it calls **recurring**$([\![p', q]\!]_k, l+1)$ (line 18). Otherwise, the algorithm changes $\overline{a}$ from p' to p by matching $\overline{p-p'}$ requests (line 14 and see Figure 1g). Consequently, it calls **recurring**$([\![p, q]\!]_k, l+1)$ (line 15). In case (♡), W_i becomes at least 1 for i in not a cyclic interval $[\![p', r]\!]_k$ (Figure 1h). Then, the algorithm calls **recurring**$([\![p', r]\!]_k, l)$ (line 18).

After completing the call of **recurring**, we have $W_i \geq 1$ for all $i \in \mathbb{Z}_k$. Then, the algorithm moves to the next phase.

To analyze the competitive ratio of this algorithm, we present several lemmas. The value of W_i changes continuously within each phase, and its rate of increase over time is given by $\mathrm{d}W_i(t)/\mathrm{d}t = \overline{s(t)-i}$. The increase rate of W_i for each i is illustrated in Figure 3.



(a) $\overline{s} \in [\![p, i-1]\!]_k$. **(b)** $\overline{s} \in [\![i, p-1]\!]_k$.

Figure 3 The increase rates $\mathrm{d}W_p/\mathrm{d}t = \overline{s-p}$ and $\mathrm{d}W_i/\mathrm{d}t = \overline{s-i}$.

We demonstrate that if W_p increases significantly while W_i increases only slightly, then W_p increases significantly during periods when $\overline{s-i}$ is small.

▶ **Lemma 10.** *Let $p, q \in \mathbb{Z}_k$ and $i \in [\![p+1, q]\!]_k$. For two times x, y $(x < y)$ in the same phase, suppose that $W_p(y) - W_p(x) \geq 2$ and $W_i(y) - W_i(x) < 1/\alpha$. Then, the increment of W_p in time t with $x \leq t \leq y$ and $\overline{s(t)} \in [\![i, p-1]\!]_k \cap [\![i, i + \lceil (\overline{i-p})/(\alpha-1) \rceil - 1]\!]_k$ is more than 1.*

Proof. Let $\beta \in \mathbb{Z}_k$ be the index such that $[\![i, \beta]\!]_k = [\![i, p-1]\!]_k \cap [\![i, i + \lceil (\overline{i-p})/(\alpha-1) \rceil - 1]\!]_k$. Define $T \coloneqq \{t \mid x \leq t \leq y \text{ and } \overline{s(t)} \in [\![i, \beta]\!]_k\}$ and $\hat{T} \coloneqq \{t \mid x \leq t \leq y \text{ and } \overline{s(t)} \notin [\![i, \beta]\!]_k\}$.

Fix $t \in \hat{T}$. Then, we have either $\overline{s(t)} \notin [\![i, p-1]\!]_k$ or $\overline{s(t)-i} \geq \lceil (\overline{i-p})/(\alpha-1) \rceil$. If $\overline{s(t)} \notin [\![i, p-1]\!]_k$, then $\frac{\mathrm{d}W_i(t)}{\mathrm{d}t} \geq \frac{\mathrm{d}W_p(t)}{\mathrm{d}t}$. Otherwise (i.e. $\overline{s(t)} \in [\![i, p-1]\!]_k$ and $\overline{s(t)-i} \geq \lceil (\overline{i-p})/(\alpha-1) \rceil$), we have

$$\frac{\mathrm{d}W_i(t)}{\mathrm{d}t} = (\overline{s(t)-i}) = \frac{(\overline{s(t)-i}) + (\alpha-1)\cdot(\overline{s(t)-i})}{\alpha} \geq \frac{(\overline{s(t)-i}) + (\overline{i-p})}{\alpha} = \frac{(\overline{s(t)-p})}{\alpha} = \frac{1}{\alpha} \cdot \frac{\mathrm{d}W_p(t)}{\mathrm{d}t}.$$

Thus, in either case, $\mathrm{d}W_p(t)/\mathrm{d}t \leq \alpha \cdot \mathrm{d}W_i(t)/\mathrm{d}t$.

Suppose to the contrary that the increment of W_p within $t \in T$ is at most 1. Under this condition, W_p must increase by at least 1 within $t \in \hat{T}$ because $W_p(y) - W_p(x) \geq 2$. Therefore, we have

$$\frac{1}{\alpha} \cdot 1 \leq \frac{1}{\alpha} \cdot \int_{\hat{T}} \frac{\mathrm{d}W_p(t)}{\mathrm{d}t}\,\mathrm{d}t \leq \int_{\hat{T}} \frac{\mathrm{d}W_i(t)}{\mathrm{d}t}\,\mathrm{d}t \leq \int_x^y \frac{\mathrm{d}W_i(t)}{\mathrm{d}t}\,\mathrm{d}t = W_i(y) - W_i(x),$$

where the second inequality holds by $\mathrm{d}W_p(t)/\mathrm{d}t \leq \alpha \cdot \mathrm{d}W_i(t)/\mathrm{d}t$ for $t \in \hat{T}$. This contradicts the assumption that $W_i(y) - W_i(x) < 1/\alpha$. Therefore, the increment of W_p within $t \in T$ is more than 1. ◀

Next, by using Lemma 10, we show that W_i increases by at least $1/\alpha$ except for some small interval after W_p increases by 2. This indicates that $\left|[\![p', q']\!]_k\right|$ is small in the situation depicted in Figure 1c.

▶ **Lemma 11.** *Let $p, q \in \mathbb{Z}_k$. For two times x, y $(x < y)$ in the same phase, suppose that $W_p(y) - W_p(x) = 2$. Then, there exist some $p', q' \in \mathbb{Z}_k$ such that $[\![p', q']\!]_k \subseteq [\![p, q]\!]_k$, $\left|[\![p', q']\!]_k\right| \leq \lceil |[\![p, q]\!]_k|/\alpha \rceil$, and $W_i(y) - W_i(x) \geq 1/\alpha$ for all $i \in [\![p, q]\!]_k \setminus [\![p', q']\!]_k$.*

Proof. If $W_i(y) - W_i(x) \geq 1/\alpha$ for all $i \in [\![p, q]\!]_k$, then the conditions are satisfied by setting $p' = q' = p$. Hence, in what follows, we assume that $W_i(y) - W_i(x) < 1/\alpha$ for some $i \in [\![p, q]\!]_k$. Define $[\![p', q']\!]_k$ $(\subseteq [\![p, q]\!]_k)$ to be the minimum cyclic interval such that $W_i(y) - W_i(x) \geq 1/\alpha$ for all $i \in [\![p, q]\!]_k \setminus [\![p', q']\!]_k$.

What is left is to show that $\left|[\![p', q']\!]_k\right| \leq \lceil |[\![p, q]\!]_k|/\alpha \rceil$. Let $\beta_i \in \mathbb{Z}_k$ be the index such that $[\![i, \beta_i]\!]_k = [\![i, p-1]\!]_k \cap [\![i, i + \lceil (\overline{i-p})/(\alpha-1) \rceil - 1]\!]_k$ and $T_i := \{t \mid x \leq t \leq y \text{ and } \overline{s(t)} \in [\![i, \beta_i]\!]_k\}$ for $i \in \{p', q'\}$. Then, by Lemma 10, the increments of W_p within $t \in T_{p'}$ and $t \in T_{q'}$ are more than 1, respectively. Since the total increment of W_p is 2, the intervals $[\![p', \beta_{p'}]\!]_k$ and $[\![q', \beta_{q'}]\!]_k$ must overlap. Thus, we have $q' \in [\![p', \beta_{p'}]\!]_k$ (see Figure 4).

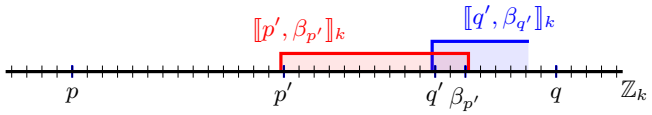


Figure 4 Relative positions of p, q, p', q', and $\beta_{p'}$ in Lemma 11.

Now, we are ready to prove $\left|[\![p', q']\!]_k\right| \leq \lceil |[\![p, q]\!]_k|/\alpha \rceil$. If $\overline{p'-p} \leq (\alpha-1) \cdot \left|[\![p, q]\!]_k\right|/\alpha$, then we have $\left|[\![p', q']\!]_k\right| \leq \left|[\![p', \beta_{p'}]\!]_k\right| \leq \lceil (\overline{p'-p})/(\alpha-1) \rceil \leq \lceil |[\![p, q]\!]_k|/\alpha \rceil$. Otherwise (i.e., $\overline{p'-p} > (\alpha-1) \cdot \left|[\![p, q]\!]_k\right|/\alpha$), we have

$$\begin{aligned}\left|[\![p', q']\!]_k\right| &\leq \left|[\![p', q]\!]_k\right| = \left|[\![p, q]\!]_k\right| - \left|[\![p, p'-1]\!]_k\right| = \left|[\![p, q]\!]_k\right| - (\overline{p'-p}) \\ &< \left|[\![p, q]\!]_k\right| - (\alpha-1) \cdot \left|[\![p, q]\!]_k\right|/\alpha = \left|[\![p, q]\!]_k\right|/\alpha \leq \lceil |[\![p, q]\!]_k|/\alpha \rceil.\end{aligned}$$

◀

Now, we demonstrate that **recurring** satisfies desired properties.

▶ **Lemma 12.** *When **recurring**$([\![p, q]\!]_k, l)$ is called, the following three conditions are satisfied: (i) $W_i \geq 1$ for all $i \notin [\![p, q]\!]_k$, (ii) $W_i \geq l/\alpha$ for all $i \in [\![p, q]\!]_k$, and (iii) $\overline{a} = p$. Moreover, if **recurring**$([\![\hat{p}, \hat{q}]\!]_k, \hat{l})$ is called next, it satisfies either (a) $|[\![\hat{p}, \hat{q}]\!]_k| \leq |[\![p, q]\!]_k|$ and $\hat{l} = l + 1$, or (b) $|[\![\hat{p}, \hat{q}]\!]_k| \leq \lceil |[\![p, q]\!]_k|/(\alpha-1) \rceil$ and $\hat{l} = l$.*

Proof. We prove these by induction.

Initially, when **recurring**$([\![0, k-1]\!]_k, 0)$ is called at the beginning of a phase, the conditions are satisfied as $W_i = 0 \geq 0/\alpha$ for all $i \in [\![0, k-1]\!]_k$ and $\overline{a} = p = 0$.

As an induction hypothesis, suppose the conditions are satisfied when **recurring**$([\![p, q]\!]_k, l)$ is called. Our goal is to show that these conditions continue to be satisfied at the next recursive call. We analyze this based on where the recursive call is made in the conditional branching of Algorithm 1. We denote by L the cardinality of $[\![p, q]\!]_k$.

If the condition of line 5 is true and recurring$([\![p, q]\!]_k, l+1)$ is called Conditions (i) and (iii) are satisfied as $\overline{a} = p$. Condition (ii) is also satisfied since the condition of line 5 is true (see Figure 1b). Moreover, the next call satisfies conditions (a).

If the condition of line 5 is false Since the proposed algorithm waits until $W_{\overline{a}}$ increases by 2 at line 2, the interval $[\![p', q']\!]_k$ chosen at line 6 satisfies $\left|[\![p', q']\!]_k\right| \leq \lceil L/\alpha \rceil$ by Lemma 11 (see Figure 1c).

If the condition of line 8 is true and recurring($[\![p, q]\!]_k, l+1$) is called Conditions (i) and (iii) are satisfied as $\overline{a} = p$. As W_p increases by $1/\alpha$ while $\overline{s} \in [\![p, p'-1]\!]_k$, W_i also increases by at least $1/\alpha$ for all $i \in [\![p', p-1]\!]_k$ by $\overline{s-i} \geq \overline{s-p}$ (see Figure 1c). Thus, for all $i \in [\![p', q']\!]_k$, W_i increases by $1/\alpha$, and hence condition (ii) is also satisfied (see Figure 1b). Additionally, the next call satisfies condition (a).

If the condition of line 8 is false Since $\overline{s} \in [\![p', p-1]\!]_k$, the proposed algorithm can match $(\overline{p'-a}) = (\overline{p'-p})$ requests at line 9 (see Figure 1d). This match results in $\overline{a} = p'$.

If the condition of line 11 is true In this scenario, **recurring**$([\![p', q]\!]_k, l+1)$ at line 13 or **recurring**$([\![p, q]\!]_k, l+1)$ at line 15 is called. In both cases, condition (a) is satisfied. In the first case, $W_{\overline{a}}$ $(= W_{p'})$ increases by 1 at line 12, which indicates $\overline{s} \in [\![p', p-1]\!]_k$ during the increase. Thus, for any $i \in [\![p, p'-1]\!]_k$, W_i increases by at least 1, and $\overline{a} = p'$ (see Figure 1f). Then, it calls **recurring**$([\![p', q]\!]_k, l+1)$ at line 13. In the second case, $\overline{s}$ is in $[\![p, p'-1]\!]_k$ and the proposed algorithm can match $(\overline{p-a}) = (\overline{p-p'})$ requests, making $\overline{a} = p$ (see Figure 1g). Then, it calls **recurring**$([\![p, q]\!]_k, l+1)$ at line 15. These calls satisfy conditions (i) and (iii) in both cases and condition (ii) is satisfied because the condition of line 11 is true.

If the condition of line 11 is false and recurring($[\![p', r]\!]_k, l$) is called There exists an index $i \in [\![p', q']\!]_k$ such that W_i increases by less than $1/\alpha$ at line 10 (Figure 1h). Let $\beta \in \mathbb{Z}_k$ be the index such that $[\![i, \beta]\!]_k = [\![i, p-1]\!]_k \cap [\![i, i + \lceil (\overline{i-p})/(\alpha-1) \rceil - 1]\!]_k$. Lemma 10 ensures that $W_{p'}$ increases by more than 1 during $\overline{s} \in [\![i, \beta]\!]_k$. Therefore, for all $j \in [\![\beta+1, p'-1]\!]_k$, W_j also increases by at least 1. As $(\overline{i-p'}) \leq (\overline{q'-p'}) = \left|[\![p', q']\!]_k\right| - 1 \leq L/\alpha$, we obtain

$$\begin{aligned}
[\![p', \beta]\!]_k &\subseteq \left[\!\left[p',\, i + \left\lceil (\overline{i-p'})/(\alpha-1) \right\rceil - 1\right]\!\right]_k \\
&\subseteq \left[\!\left[p',\, p' + (\overline{i-p'}) + \left\lceil \tfrac{1}{\alpha-1} \cdot (\overline{i-p'}) \right\rceil - 1\right]\!\right]_k \\
&= \left[\!\left[p',\, p' + \left\lceil \tfrac{\alpha}{\alpha-1} \cdot (\overline{i-p'}) \right\rceil - 1\right]\!\right]_k \subseteq \left[\!\left[p',\, p' + \lceil L/(\alpha-1) \rceil - 1\right]\!\right]_k.
\end{aligned}$$

As $[\![p', r]\!]_k \subseteq [\![p', \beta]\!]_k$, we have $|[\![p', r]\!]_k| \leq \lceil L/(\alpha-1) \rceil$ and condition (b) is satisfied. Since $\overline{a} = p'$, conditions (i) and (iii) are satisfied, and by the induction hypothesis, condition (ii) is also satisfied. ◀

We can bound the number of recursions called by the proposed algorithm from Lemma 12, giving us the upper bound of the cost incurred by the proposed algorithm during each phase.

▶ **Lemma 13.** *The proposed algorithm incurs a cost of $O(\alpha)$ per phase.*

Proof. It is not difficult to see that the cost incurred by the algorithm in each recursive call is no more than 8. Thus, it is sufficient to prove that the number of recursions is at most $O(\alpha)$ in each phase.

As our goal is asymptotic evaluation, we may assume that $\alpha \geq 4$, which implies $\alpha \leq (\alpha-1)^2$. By Lemma 12, each call of **recurring**$([\![p, q]\!]_k, l)$ either increments l by 1 or reduces the number of elements in $[\![p, q]\!]_k$ to at most $\lceil |[\![p, q]\!]_k|/(\alpha-1) \rceil$. Recall that the recursion starts with $|[\![p, q]\!]_k| = k$ $(= \alpha^\alpha)$ and $l = 0$ and ends when $|[\![p, q]\!]_k| = 1$ or $l \geq \alpha$.

Let L_n be the number of elements in $[\![p, q]\!]_k$ after the interval reduction has occurred n times. Then, we have $L_n \leq \lceil L_{n-1}/(\alpha-1) \rceil \leq L_{n-1}/(\alpha-1) + 1$, which implies

$$\left(L_n - \tfrac{\alpha-1}{\alpha-2}\right) \leq \frac{1}{\alpha-1} \cdot \left(L_{n-1} - \tfrac{\alpha-1}{\alpha-2}\right) \leq \cdots \leq \frac{1}{(\alpha-1)^n} \cdot \left(L_0 - \tfrac{\alpha-1}{\alpha-2}\right) \leq \frac{L_0}{(\alpha-1)^n}.$$

Thus, the number of elements in the interval after $2\lceil\alpha\rceil + 1$ iterations is at most

$$L_{2\lceil\alpha\rceil+1} \leq \frac{\alpha^\alpha}{(\alpha-1)^{2\alpha+1}} + \frac{\alpha-1}{\alpha-2} \leq \frac{1}{\alpha-1} + \frac{\alpha-1}{\alpha-2} = 1 + \frac{1}{\alpha-1} + \frac{1}{\alpha-2} < 2,$$

where the second inequality holds by $\alpha \leq (\alpha-1)^2$ and the third inequality holds by $1/(\alpha-1) < 1/(\alpha-2) \leq 1/2$ from the assumption that $\alpha \geq 4$. Consequently, the number of elements in $[\![p,q]\!]_k$ is reduced to 1 or l increases to $\lceil\alpha\rceil$ in at most $3\lceil\alpha\rceil = O(\alpha)$ iterations. Thus, the number of recursive calls is $O(\alpha)$. ◀

Next, we provide a lower bound on the cost incurred by any algorithm during each phase.

▶ **Lemma 14.** *At the end of a phase, the cost incurred within the phase by any algorithm is at least* 1 *(if we treat the waiting costs as imposed sequentially at each moment, rather than at the time of matching).*

Proof. Recall that, if $(k-i)$ unmatched requests are carried over to a phase, the optimum cost incurred thus far within the phase is at least $\min\{1, W_i\}$. Thus, it is sufficient to prove that $W_i \geq 1$ for all $i \in \mathbb{Z}_k$ at the end of the phase.

The phase ends at line 4. At this point, we have $\left|[\![p,q]\!]_k\right| = 1$ or $l \geq \alpha$, according to the condition at line 3. If $\left|[\![p,q]\!]_k\right| = 1$, then we have $p = q$, and $W_i \geq 1$ for all $i \neq p$ by Lemma 12. Since $W_p \geq 1$ due to the wait performed at line 2, it follows that $W_i \geq 1$ for all $i \in \mathbb{Z}_k$. Conversely, if $l \geq \alpha$, we have $W_i \geq 1$ for all $i \notin [\![p,q]\!]_k$ and $W_i \geq l/\alpha \geq 1$ for all $i \in [\![p,q]\!]_k$ by Lemma 12. Thus, in either case, we obtain $W_i \geq 1$ for all $i \in \mathbb{Z}_k$ at the end of the phase. This proves the lemma. ◀

Based on the lemmas above, we now prove Theorem 8.

Proof of Theorem 8. Suppose that the proposed algorithm completes p (≥ 1) phases for the given instance. Then, the cost for the proposed algorithm is at most $(p+1)\cdot O(\alpha)$ by Lemma 13, while the cost for any algorithm is at least p by Lemma 14. Therefore, the competitive ratio for this instance is at most $(p+1)\cdot O(\alpha)/p = O(\alpha)$.

Conversely, suppose that the instance ends during the first phase. If the optimal offline algorithm incurs a cost of at least 1, then the competitive ratio is at most $O(\alpha)$, as the cost incurred by the proposed algorithm during the first phase is $O(\alpha)$. If the cost incurred by the optimal offline algorithm is less than 1, then only waiting costs are incurred, as no requests are carried over to the first phase. This means that the total cost of the optimal offline algorithm is equal to W_0. Since the proposed algorithm continues to wait until W_0 reaches 2 at line 2, the total cost of the proposed algorithm is also equal to W_0, resulting in a competitive ratio of 1 for such an instance.

Therefore, the proposed algorithm is $O(\alpha)$ $(= O(\log k/\log\log k))$-competitive. ◀

4.2 Lower Bound

In this subsection, we prove the following lower bound of the competitive ratio.

▶ **Theorem 15.** *For the OMDSC problem with a penalty function f that satisfies $B_f = \{kn \mid n \in \mathbb{Z}_{++}\}$, every deterministic online algorithm has a competitive ratio of $\Omega(\log k/\log\log k)$.*

Recall that α is a positive real such that $\alpha^\alpha = k$. We fix an arbitrary online algorithm ALG and construct an adversarial instance according to the behavior of ALG, denoted as σ_{ALG}. The instance σ_{ALG} is composed of $\Theta(\alpha)$ rounds. To construct σ_{ALG}, we introduce variables similar to Definition 9.

▶ **Definition 16.** *For the instance σ_{ALG} and time t, the variables $W_0(t), W_1(t), \ldots, W_{k-1}(t)$, $s(t)$, and $a(t)$ are defined as follows:*

- *$W_i(t)$ $(i \in \mathbb{Z}_k)$: the waiting cost incurred by time t for an algorithm that initially matches $(\overline{i-1})$ requests (i.e, leaving $(\overline{k-i})$ requests) at time 0 and subsequently performs matches of size k immediately for every k unmatched requests accumulated.*

- $s(t)$: *the number of requests given by time* t.
- $a(t)$: *the number of requests matched by* ALG *up to, but not including, time* t.

We will omit the argument t when there is no confusion.

The instance σ_{ALG} contains $k-1$ requests with arrival time at 0. For σ_{ALG} and any $i \in \mathbb{Z}_k$, there exists an offline algorithm that incurs a cost of at most $2 + W_i(x_{n^*})$, where x_{n^*} is the time when the last requests are given in σ_{ALG}. This can be achieved by matching $(\overline{i-1})$ requests at time 0, greedily matching a multiple of k requests in the middle, and matching all the remaining requests at time x_{n^*}. For each round n, the starting time is x_n, and there is a (non-empty) cyclic interval $[\![p_n, q_n]\!]_k$ such that $W_i(x_n) \le 2\alpha/n$ for every $i \in [\![p_n, q_n]\!]_k$. As rounds progress, the interval is gradually narrowed, and ALG incurs a cost of $\Omega(1)$ per round. The instance σ_{ALG} consists of $\Theta(\alpha)$ rounds. This implies that while there exists an offline algorithm with a cost of $O(1)$, ALG incurs a cost of $\Omega(\alpha)$, indicating that the competitive ratio of ALG is $\Omega(\alpha) = \Omega(\log k / \log\log k)$.

Initially, the instance gives $k-1$ requests at time 0 (i.e. $\overline{a(0)} = 0$ and $\overline{s(0)} = k-1$). Let n be the variable that indicates the round, initialized to 0. Define $p_0 = 0$, $q_0 = k-1$, and $x_0 = 0$.

For the nth round ($n \ge 0$), if $\left|[\![p_n, q_n]\!]_k\right| < \alpha^2$, set n^* to be n and finalize the instance. Otherwise, the round continues until time $x_{n+1} = x_n + 1/(\overline{s(x_n) - a(x_n)})$. Requests in round n are provided at time x_{n+1} based on the number of remaining requests immediately before time x_{n+1} (i.e., $\overline{s(x_n) - a(x_{n+1})}$).

Let h_n be $\left\lceil |[\![p_n, q_n]\!]_k| / \alpha^2 \right\rceil$. If $\overline{a(x_{n+1})} \notin [\![q_n - h_n + 1, q_n]\!]_k$, then set (p_{n+1}, q_{n+1}) to be $(\overline{q_n - h_n + 1}, q_n)$ and give no requests. Otherwise, let $(p_{n+1}, q_{n+1}) = (\overline{q_n - 2h_n + 1}, \overline{q_n - h_n})$, and give $\overline{q_{n+1} - q_n}$ requests so that $\overline{s(x_{n+1})} = q_{n+1}$ at time x_{n+1} (see Figure 5). Then, increase round count n by 1 and proceed to the next round.

This instance ensures that (i) $k/\alpha^{2n} \le |[\![p_n, q_n]\!]_k| \le k/\alpha^n$, (ii) $W_i(x_n) \le 2n/\alpha$ for all $i \in [\![p_n, q_n]\!]_k$, and (iii) $\overline{a(x_n)} \in [\![q_n + 1, p_n]\!]_k$ and $\overline{s(x_n)} = q_n$.

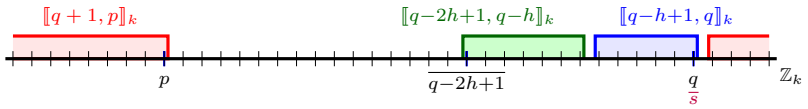


Figure 5 Relative positions of the cyclic intervals.

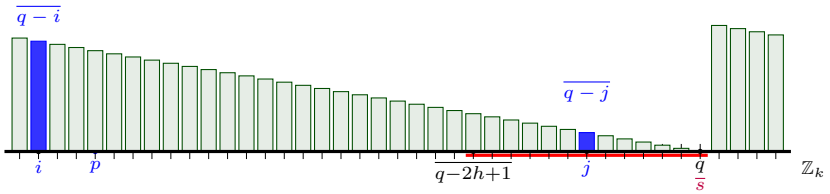


Figure 6 Increment rates $\mathrm{d}W_i/\mathrm{d}t = \overline{q-i}$ and $\mathrm{d}W_j/\mathrm{d}t = \overline{q-j}$.

Now, we show a lemma that establishes a lower bound on the number of elements in a cyclic interval where the increment of W_i in the interval is at most $2/\alpha$ times the increment of W_j for specific j.

▶ **Lemma 17.** *For a cyclic interval* $[\![p, q]\!]_k$, *let* $L \coloneqq \left|[\![p, q]\!]_k\right|$ *and* $h \coloneqq \left\lceil L/\alpha^2 \right\rceil$. *Suppose that* $L \ge \alpha^2$ *and* $\alpha \ge 4$. *Then, for any* $i \in [\![q+1, p]\!]_k$ *and* $j \in [\![q-2h+1, q]\!]_k$, *the increment of* W_j *is at most* $2/\alpha$ *times the increment of* W_i *in the state where* $\overline{s} = q$.

Proof. Figure 6 depicts the increment rates $\mathrm{d}W_i/\mathrm{d}t = \overline{q-i}$ and $\mathrm{d}W_j/\mathrm{d}t = \overline{q-j}$. By the assumption that $L \geq \alpha^2$ and $\alpha \geq 4$, we have

$$\frac{2}{\alpha}\cdot(L-1) - \left(\frac{2L}{\alpha^2}+1\right) = \frac{2L}{\alpha}\cdot\left(1-\frac{1}{\alpha}\right) - \frac{2}{\alpha} - 1 \geq \frac{2\alpha^2}{\alpha}\cdot\left(1-\frac{1}{4}\right) - \frac{2}{4} - 1 > 0. \tag{1}$$

Therefore, for any $i \in [\![q+1, p]\!]_k$ and $j \in [\![q-2h+1, q]\!]_k$, we get

$$\frac{\mathrm{d}W_j}{\mathrm{d}t} = \overline{q-j} \leq 2\cdot\left\lceil\frac{L}{\alpha^2}\right\rceil - 1 \leq 2\cdot\frac{L}{\alpha^2}+1 \leq \frac{2}{\alpha}\cdot(L-1) = \frac{2}{\alpha}\cdot\overline{q-p} = \frac{2}{\alpha}\cdot\frac{\mathrm{d}W_i}{\mathrm{d}t},$$

where the third inequality follows from (1). Since the rate of increase is constant while $\overline{s} = q$, the increment of W_j is at most $2/\alpha$ times the increment of W_i. ◀

By using this lemma, we bound the number of elements in $[\![p, q]\!]_k$ after n rounds from below and bound the values of W_i for all $i \in [\![p, q]\!]_k$ from above.

▶ **Lemma 18.** *Let $n \in \{0, 1, 2, \ldots, n^*\}$ and suppose that $\alpha \geq 4$. At time x_n, the following three conditions are satisfied: (i) $k/\alpha^{2n} \leq \left|[\![p_n, q_n]\!]_k\right| \leq k/\alpha^n$, (ii) $W_i(x_n) \leq 2n/\alpha$ for all $i \in [\![p_n, q_n]\!]_k$, and (iii) $\overline{a(x_n)} \in [\![q_n+1, p_n]\!]_k$ and $\overline{s(x_n)} = q_n$.*

Proof. We prove this by induction on n.

At time $x_0 = 0$, we have $[\![p_0, q_0]\!]_k = [\![0, k-1]\!]_k = k = k/\alpha^0$, and $W_i(x_0) = 0 \leq 2\cdot 0/\alpha$ for all $i \in [\![0, k-1]\!]_k = [\![p_0, q_0]\!]_k$. In addition, we have $\overline{a(x_0)} = 0 \in [\![0, 0]\!]_k = [\![k, 0]\!]_k$ and $\overline{s(x_0)} = k-1$. Thus, conditions (i), (ii), and (iii) are satisfied.

Let $n' \in \{0, 1, 2, \ldots, n^*-1\}$. Suppose that conditions (i), (ii), and (iii) are satisfied for $n = n'$. We show that these conditions are also satisfied for $n = n'+1$.

Recall that $h_n = \left\lceil\left|[\![p_n, q_n]\!]_k\right|/\alpha^2\right\rceil$. Then, there are two cases where (p_{n+1}, q_{n+1}) is set to be $(\overline{q_n - h_n + 1}, q_n)$ or $(\overline{q_n - 2h_n + 1}, \overline{q_n - h_n})$. In both cases, we prove the inductive step by using Lemma 17. Note that $\left|[\![p_{n+1}, q_{n+1}]\!]_k\right| = h_n$, $\overline{a(x_{n+1})} \notin [\![p_{n+1}, q_{n+1}]\!]_k$ and $\overline{s(x_{n+1})} = q_{n+1}$ in both cases by the definition of the procedure. Thus, the condition (iii) is met. By the induction hypothesis, we have $k/\alpha^{2n} \leq \left|[\![p_n, q_n]\!]_k\right| \leq k/\alpha^n$ and $\overline{a(x_n)} \in [\![q_n+1, p_n]\!]_k$. Also, as $n < n^*$, we have $\left|[\![p_n, q_n]\!]_k\right| \geq \alpha^2$. Thus, we have

$$\left|[\![p_{n+1}, q_{n+1}]\!]_k\right| = \left\lceil\frac{\left|[\![p_n, q_n]\!]_k\right|}{\alpha^2}\right\rceil \geq \left\lceil\frac{k/\alpha^{2n}}{\alpha^2}\right\rceil \geq \frac{k}{\alpha^{2n+2}}.$$

Also, we have

$$\left|[\![p_{n+1}, q_{n+1}]\!]_k\right| = \left\lceil\frac{\left|[\![p_n, q_n]\!]_k\right|}{\alpha^2}\right\rceil \leq 1 + \frac{\left|[\![p_n, q_n]\!]_k\right|}{\alpha^2} \leq \frac{2\left|[\![p_n, q_n]\!]_k\right|}{\alpha^2} \leq \frac{\left|[\![p_n, q_n]\!]_k\right|}{\alpha} \leq \frac{k}{\alpha^{n+1}},$$

where the second inequality holds by $\left|[\![p_n, q_n]\!]_k\right| \geq \alpha^2$ and the third inequality holds by $\alpha \geq 4$. Thus, the condition (i) is satisfied in both cases. Moreover, we have

$$W_{p_n}(x_{n+1}) - W_{p_n}(x_n) = (x_{n+1} - x_n)\cdot(\overline{s(x_n) - p_n}) = \frac{(\overline{s(x_n)-p_n})}{(\overline{s(x_n)-a(x_n)})} \leq 1,$$

because $\overline{s(x_n)} = q_n$ and $\overline{a(x_n)} \in [\![q_n+1, p_n]\!]_k$ (see Figure 6). Thus, the increment of W_j is less than $2/\alpha$ for all $j \in [\![p_{n+1}, q_{n+1}]\!]_k \subseteq [\![q_n - 2h_n + 1, q_n]\!]_k$ by Lemma 17 with setting $(p, q) = (p_n, q_n)$ and $i = p_n$. Therefore, combining with the induction hypothesis, we obtain $W_j(x_{n+1}) \leq 2(n+1)/\alpha$ for all $j \in [\![p_{n+1}, q_{n+1}]\!]_k$. This means that the condition (ii) is satisfied.

Therefore, the conditions (i), (ii), and (iii) are satisfied for $n = n'+1$. This completes the proof by induction. ◀

In what follows, we bound the cost incurred by any online algorithm ALG on σ_{ALG} is at least $\Omega(\alpha)$ and the cost for the optimal offline algorithm on σ_{ALG} is at most a constant. As our goal is asymptotic evaluation, we assume that $\alpha \geq 4$ in the following.

▶ **Lemma 19.** *The number of rounds n^* is $\Theta(\alpha)$.*

Proof. For $n \in \{0, 1, 2, \ldots, n^*\}$, let $L_n = \left|[\![p_n, q_n]\!]_k\right|$. From Lemma 18, we have $k/\alpha^{2n} \leq L_n \leq k/\alpha^n$. By the termination condition, we have $L_{n^*} < \alpha^2$ and $L_{n^*-1} \geq \alpha^2$. As $\alpha^2 > L_{n^*} \geq k/\alpha^{2n^*} = \alpha^{\alpha-2n^*}$, we have $n^* \geq (\alpha-2)/2 = \alpha/2 - 1$. In addition, as $\alpha^2 \leq L_{n^*-1} \leq k/\alpha^{n^*-1} = \alpha^{\alpha-n^*+1}$, we have $n^* \leq \alpha - 1 \leq \alpha$. Therefore, we obtain $\alpha/2 - 1 \leq n^* \leq \alpha$, and hence $n^* = \Theta(\alpha)$. ◀

From Lemma 19, we can prove the following lemmas.

▶ **Lemma 20.** *For any deterministic online algorithm* ALG*, there exists an offline algorithm that incurs a cost of at most a constant for the instance σ_{ALG}.*

Proof. Let i^* be an arbitrary element in $[\![p_{n^*}, q_{n^*}]\!]_k$. Consider the algorithm that initially matches $(\overline{i^*-1})$ requests at time 0, subsequently performs matches of size k immediately for every k unmatched requests accumulated, and finally matches all the remaining unmatched requests at x_{n^*}. Then, the total waiting cost of this algorithm is at most $W_{i^*}(x_{n^*}) \leq 2n^*/\alpha \leq 2$ by Lemmas 18 and 19. Moreover, the total size cost is at most 2, as it matches at most twice with sets of requests of size not a multiple of k. Therefore, The total cost incurred by this algorithm is at most 4. ◀

▶ **Lemma 21.** $\mathrm{ALG}(\sigma_{\mathrm{ALG}}) = \Omega(\alpha)$ *for any deterministic online algorithm* ALG*.*

Proof. For the nth round, we divide into two cases in which ALG matches a non-multiple of k requests during a period $[x_n, x_{n+1})$ or not. If ALG matches a non-multiple of k requests, it incurs a size cost of 1. Otherwise, $\overline{a(t)} = \overline{a(x_n)}$ for all $t \in [x_n, x_{n+1})$, leading to that the increment of $W_{\overline{a(x_n)}}$ during $[x_n, x_{n+1})$ is 1 because $x_{n+1} - x_n = 1/(\overline{s(x_n) - a(x_n)})$ and its increase rate is $(\overline{s(x_n) - a(x_n)})$. In both cases, the algorithm incurs at least a cost of 1 in each round.

Therefore, the total cost incurred over the instance is at least $1 \cdot n^* = \Omega(\alpha)$ by Lemma 19. ◀

Now, we are ready to prove Theorem 15.

Proof of Theorem 15. By combining Lemmas 20 and 21, the competitive ratio of ALG is at least

$$\sup_{\sigma} \frac{\mathrm{ALG}(\sigma)}{\mathrm{OPT}(\sigma)} \geq \frac{\mathrm{ALG}(\sigma_{\mathrm{ALG}})}{\mathrm{OPT}(\sigma_{\mathrm{ALG}})} \geq \frac{\Omega(\alpha)}{O(1)} = \Omega(\alpha) = \Omega\left(\frac{\log k}{\log\log k}\right). \qquad ◀$$

References

1 Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 1051–1061, 2017. doi:10.1137/1.9781611974782.67.

2 Yossi Azar, Ashish Chiplunkar, Shay Kutten, and Noam Touitou. Set cover with delay – Clairvoyance is not required. In *Proceedings of the 28th Annual European Symposium on Algorithms (ESA 2020)*, pages 8:1–8:21, 2020. doi:10.4230/LIPIcs.ESA.2020.8.

3 Yossi Azar and Amit Jacob Fanani. Deterministic min-cost matching with delays. *Theory Comput. Syst.*, 64(4):572–592, 2020. doi:10.1007/s00224-019-09963-7.

4 Yossi Azar, Runtian Ren, and Danny Vainstein. The min-cost matching with concave delays problem. In *Proceedings of the 2021 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 301–320, 2021. doi:10.1137/1.9781611976465.20.

5 Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *Proceedings of the 2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS 2019)*, pages 60–71, 2019. doi:10.1109/FOCS.2019.00013.

6 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Luk'avs Folwarczn'y, Lukasz Jez, Jiri Sgall, Kim Thang Nguyen, and Pavel Veselý. Online algorithms for multi-level aggregation. In *Proceedings of the 24th Annual European Symposium on Algorithms (ESA 2016)*, pages 12:1–12:17, 2016. doi:10.4230/LIPIcs.ESA.2016.12.

7 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Lukasz Jez, Dorian Nogneng, and Jirí Sgall. Better approximation bounds for the joint replenishment problem. In *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 42–54, 2014. doi:10.1137/1.9781611973402.4.

8 Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Pawel Schmidt. A primal-dual online deterministic algorithm for matching with delays. In *Proceedings of the 16th Workshop on Approximation and Online Algorithms (WAOA 2018)*, pages 51–68, 2018. doi:10.1007/978-3-030-04693-4_4.

9 Niv Buchbinder, Moran Feldman, Joseph S. Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 1235–1244, 2017. doi:10.1137/1.9781611974782.80.

10 Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online make-to-order joint replenishment model: Primal-dual competitive algorithms. *Oper. Res.*, 61(4):1014–1029, 2013. doi:10.1287/opre.2013.1188.

11 Rodrigo A. Carrasco, Kirk Pruhs, Cliff Stein, and José Verschae. The online set aggregation problem. In *Proceedings of the 13th Latin American Symposium on Theoretical Informatics (LATIN 2018)*, pages 245–259, 2018. doi:10.1007/978-3-319-77404-6_19.

12 Ryder Chen, Jahanvi Khatkar, and Seeun William Umboh. Online weighted cardinality joint replenishment problem with delay. In *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, pages 40:1–40:18, 2022. doi:10.4230/LIPIcs.ICALP.2022.40.

13 Lindsey Deryckere and Seeun William Umboh. Online matching with set and concave delays. In *Proceedings of the Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*, pages 17:1–17:17, 2023. doi:10.4230/LIPIcs.APPROX/RANDOM.2023.17.

14 Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. On-line analysis of the TCP acknowledgment delay problem. *J. ACM*, 48(2):243–273, 2001. doi:10.1145/375827.375843.

15 Federico Echenique, Nicole Immorlica, and Vijay V. Vazirani. *Online and Matching-Based Market Design*. Cambridge University Press, 2023. doi:10.1017/9781108937535.

16 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: Haste makes waste! In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing (STOC 2016)*, pages 333–344, 2016. doi:10.1145/2897518.2897557.

17 Yuval Emek, Yaacov Shapiro, and Yuyi Wang. Minimum cost perfect matching with delays for two sources. *Theor. Comput. Sci.*, 754:122–129, 2019. doi:10.1016/j.tcs.2018.07.004.

18 Kun He, Sizhe Li, Enze Sun, Yuyi Wang, Roger Wattenhofer, and Weihao Zhu. Randomized algorithm for MPMD on two sources. In *Proceedings of the 19th Conference On Web And InterNet Economics (WINE 2023)*, pages 348–365, 2023. doi:10.1007/978-3-031-48974-7_20.

19 Naonori Kakimura and Tomohiro Nakayoshi. Deterministic primal-dual algorithms for online k-way matching with delays. In *Proceedings of the 29th International Computing and Combinatorics Conference (COCOON 2023)*, pages 238–249, 2023. doi:10.1007/978-3-031-49193-1_18.

20 Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgement and other stories about $e/(e-1)$. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC 2001)*, pages 502–509, 2001. doi:10.1145/380752.380845.

21 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC 1990)*, pages 352–358, 1990. doi:10.1145/100216.100262.

22 Yasushi Kawase and Tomohiro Nakayoshi. Online matching with delays and size-based costs. *arXiv:2408.08658*, 2024. doi:10.48550/arXiv.2408.08658.

23 Xingwu Liu, Zhida Pan, Yuyi Wang, and Roger Wattenhofer. Impatient online matching. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC 2018)*, pages 62:1–62:12, 2018. doi:10.4230/LIPIcs.ISAAC.2018.62.

24 Xingwu Liu, Zhida Pan, Yuyi Wang, and Roger Wattenhofer. Online matching with convex delay costs. *arXiv:2203.03335*, 2022. doi:10.48550/arXiv.2203.03335.

25 Aranyak Mehta. Online matching and ad allocation. *Found. Trends Theor. Comput. Sci.*, 8(4):265–368, 2013. doi:10.1561/0400000057.

26 Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. AdWords and generalized online matching. *J. ACM*, 54(5):22:1–22:19, 2007. doi:10.1145/1284320.1284321.

27 Darya Melnyk, Yuyi Wang, and Roger Wattenhofer. Online k-way matching with delays and the H-metric. *arXiv:2109.06640*, 2021. doi:10.48550/arXiv.2109.06640.

28 Marco Pavone, Amin Saberi, Maximilian Schiffer, and Matthew W. Tsao. Technical note—online hypergraph matching with delays. *Oper. Res.*, 70(4):2194–2212, 2022. doi:10.1287/opre.2022.2277.

29 Noam Touitou. Nearly-tight lower bounds for set cover and network design with deadlines/delay. In *Proceedings of the 32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, pages 53:1–53:16, 2021. doi:10.4230/LIPIcs.ISAAC.2021.53.