Online Matching with Set and Concave Delays

Lindsey Deryckere
School of Computer Science, The University of Sydney, Australia

Seeun William Umboh
School of Computing and Information Systems, The University of Melbourne, Australia

Abstract

We initiate the study of online problems with *set delay*, where the delay cost at any given time is an arbitrary function of the set of pending requests. In particular, we study the online min-cost perfect matching with set delay (MPMD-Set) problem, which generalises the online min-cost perfect matching with delay (MPMD) problem introduced by Emek et al. (STOC 2016). In MPMD, m requests arrive over time in a metric space of n points. When a request arrives the algorithm must choose to either match or delay the request. The goal is to create a perfect matching of all requests while minimising the sum of distances between matched requests, and the total delay costs incurred by each of the requests. In contrast to previous work we study MPMD-Set in the *non-clairvoyant* setting, where the algorithm does not know the future delay costs. We first show no algorithm is competitive in n or m. We then study the natural special case of *size-based* delay where the delay is a non-decreasing function of the number of unmatched requests. Our main result is the first non-clairvoyant algorithms for online min-cost perfect matching with size-based delay that are competitive in terms of m. In fact, these are the first non-clairvoyant algorithms for any variant of MPMD. A key technical ingredient is an analog of the symmetric difference of matchings that may be useful for other special classes of set delay. Furthermore, we prove a lower bound of $\Omega(n)$ for any deterministic algorithm and $\Omega(\log n)$ for any randomised algorithm. These lower bounds also hold for clairvoyant algorithms. Finally, we also give an m-competitive deterministic algorithm for uniform concave delays in the clairvoyant setting.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online algorithms, matching, delay, non-clairvoyant

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2023.17

Category APPROX

Related Version *Full Version*: `https://arxiv.org/abs/2211.02394`

1 Introduction

Studying online problems with delay is a line of work that has recently gained traction in online algorithms (e.g. [4, 19, 21, 23]). In such problems, request arrive over time requiring service. Delaying the service of a request accumulates a delay cost given by a delay function associated with the request. The total cost of a solution is the cost of servicing all requests plus the sum of all delay costs incurred by each request.

We initiate the study of online problems with *set delay*. In this model, we generalize the notion of delay to one where the instantaneous delay cost at any point in time is determined by an arbitrary monotone non-decreasing function of the set of pending requests, rather than the sum of individual delay functions associated with each request. In particular, we study the online min-cost perfect matching with set delay (MPMD-Set) problem, which generalizes of the min-cost perfect matching with delays (MPMD) problem introduced by Emek et al. [19].

In MPMD, m requests arrive over time in a metric space of n points. Upon arrival of a request the algorithm must choose to either match the request, incurring a cost equal to the distance between the two requests, or to delay the request, incurring a cost given by a delay function associated with the request. Prior results for MPMD have mostly focused on each request sharing the same delay function (in particular, linear, concave, and convex) and achieve competitive ratios that solely depend either on n or m. Moreover, existing algorithms rely on clairvoyance, where the algorithm has full knowledge of future delay costs. Furthermore, existing randomised algorithms rely on metric embeddings which require knowledge of the metric space in advance.

In this paper, our main contribution is to study the more general MPMD-Set in the least restrictive setting where the algorithm does not know the metric space in advance and has no knowledge of future delay costs. We begin by showing that, in contrast to prior results, the MPMD-Set problem does not admit a deterministic competitive ratio that solely depends on n or m.

▶ **Theorem 1.** *Every deterministic algorithm for MPMD-Set has competitive ratio $\Omega(\Phi)$, where Φ is the aspect ratio of the metric space.*

Our lower bound holds even for simple instances where n and m are constants. Thus, we restrict our attention to designing a competitive solution for the MPMD-Set problem where the instantaneous delay cost at any point in time is a monotone non-decreasing function of the number of unmatched requests at that time. We call such a delay cost function *size-based* (See Section 2 for a formal definition). MPMD-Set with size-based delay (MPMD-Size) has natural applications in practical settings with service-level agreements such as cloud computing.[1]

Our main result is the first competitive algorithms for MPMD-Size, where the competitive ratio is a function of the number of requests. At the core of our result is a reduction from MPMD-Size to the well-known Metrical Task System (MTS) problem (defined in Section 1.1).

▶ **Theorem 2.** *For any $f(N)$-competitive algorithm for MTS with N states, there is an $f(2^m)$-competitive algorithm for MPMD-Size.*

We obtain our main result by applying state-of-the-art algorithms for MTS with some modifications.

▶ **Corollary 3.** *For MPMD-Size, there is an $O(2^m)$-competitive deterministic algorithm and an $O(m^4)$-competitive randomised algorithm.*

We emphasise that our algorithms are non-clairvoyant and do not need to know the metric space in advance. To the best of our knowledge, this is the first non-clairvoyant online algorithm for this problem. Non-clairvoyant algorithms nevertheless have been designed for other online problems such as the Set Cover problem [4], the k-server problem [25], and multi-level aggregation [26]. We also remark that every deterministic algorithm for known variants of online matching with delays has a competitive ratio that depends on m.

We complement Corollary 3 with the following lower bounds.

▶ **Theorem 4.** *Every deterministic algorithm for MPMD-Size has competitive ratio $\Omega(n)$.*

▶ **Theorem 5.** *Every randomised algorithm for MPMD-Size has competitive ratio $\Omega(\log n)$.*

[1] In these settings, the service level agreement requires the cloud provider to provide a certain level of service and the provider incurs penalties if the level is not met.

Finally, we consider MPMD with uniform concave delay in the clairvoyant setting and give the first deterministic algorithm for it. In this problem, we are given a non-negative, non-decreasing concave function f. The delay cost incurred by a request r is $f(w_r)$ where w_r is the time between r's arrival and when it was matched. The total delay cost is the sum of the delay cost of each request.

▶ **Theorem 6.** *There exists an $O(m)$-competitive deterministic algorithm for MPMD with concave delay.*

The correctness and competitiveness of our algorithm only relies on the fact that the time-augmented space satisfies the properties of a metric space. Similar to previous deterministic solutions for uniform linear delay, our algorithm does not need the metric space to be finite, and does not need to know it in advance.

The proofs of theorems 4, 5 and 6 can be found in the full version.

1.1 Our Techniques

Our main technical contribution is an online reduction from the MPMD-Set problem to MTS, which constitutes the proof of Theorem 2. The Metrical Task System (MTS) problem, introduced by Borodin et al. [15], is a cost minimisation problem defined by a set of states $S = \{s_1, s_2, ..., s_k\}$ and a cost matrix c that defines the cost of moving between states. The input consists of an initial state S_0 and a sequence of tasks $T = (t_1, ..., t_\ell)$. Each task t_j is associated with a k-dimensional cost vector C_j whose i-th coordinate defines the cost of servicing task t_j in state s_i. For a given input task sequence T, a solution is a sequence of states (called a *schedule*) $\sigma = (S_1, S_2, ..., S_\ell)$, where S_j is the state that task j is processed in. The total cost of a schedule consists of the costs associated with moving states (*transition cost*), as well as the cost of processing the tasks (*processing cost*).The aim is to produce a schedule of minimum cost.

We briefly outline the three main parts of the reduction below.

Step 1: MPMD-Set to MTS. The first part of the reduction transforms an instance of MPMD-Set into an instance of MTS. A natural approach at a reduction to MTS is to use the set of all possible matchings of requests as the set of states for the MTS instance. The transition cost between two states is then the total length of the edges in the symmetric difference of the corresponding matchings. Finally, there is a task for each timestep in the matching problem and the cost of processing the task in a state is the instantaneous delay incurred by the set of unmatched requests. Unfortunately, the number of states is equal to the number of possible matchings between the requests which is $\sim (\frac{m}{e})^{m/2} \frac{e^{\sqrt{m}}}{(4e)^{1/4}}$ for m requests.

Instead, we use the set of all possible even-sized subsets of the requests as the set of MTS states. Each state represents a set of requests that are matched. The set of input states thus develops over time as more requests arrive. The initial state is the empty set.

We now define the transition costs of the MTS. We define a *transition graph* $G(V, E)$ where V is the set of even-sized subsets of requests. The transition graph G has an edge between two states S and S' if $S \subset S'$ and $|S'| = |S| + 2$. In other words, S_2 consists of the same requests as S_1 with 2 additional requests, say p, q. The cost of the edge between the two states is $d(p, q)$, the distance between p and q in the original MPMD-Set instance. The transition cost between any two states in the MTS instance is defined to be the minimum cost path between the two corresponding nodes in G. The delay cost is translated into the vector costs associated with serving tasks: for any timestep t, the cost of servicing a task in state is simply the instantaneous delay cost accumulated by the set of requests that have arrived so far in the original MPMD-Set instance, that are not in that state.

Henceforth, we refer to an instance of MTS that is reduced from MPMD-Set as *MPMD-Set-MTS*, and an MTS instance that is reduced from MPMD-Set with size-based delay as *MPMD-Size-MTS*.

▶ **Definition 7** (Monotone). *A schedule $\sigma = (S_1, \ldots, S_\ell)$ is* monotone *if every transition only adds requests to the current state, i.e. $S_{i-1} \subseteq S_i$ for every i. In other words, the path never involves moving to a strictly smaller state. An algorithm for MPMD-Set-MTS is* monotone *if it always produces a* monotone *schedule.*

It is easy to see that a monotone schedule can be converted into a solution for the MPMD-Set instance without any increase in cost: when the schedule adds a set of requests S' to its current state, we add a min-cost perfect matching on S' to our matching. However, when we run existing MTS algorithms on the MPMD-Set-MTS instance, there is no guarantee that they will return a monotone schedule.

Step 2: Converting to Monotone. Fortunately, it is possible to convert, in an online manner, an arbitrary MPMD-Size-MTS solution to a monotone solution at no extra cost. We do this by designing an online algorithm which, given an online sequence of states $\sigma = (S_1, \ldots, S_\ell)$, produces for each state S_i a corresponding state S'_i such that the resulting schedule produced by the algorithm is *monotone*. We refer to an algorithm that *transforms* a given state as a *state conversion algorithm*.

Since the instantaneous delay in the MPMD-Size instance is a non-increasing function of currently matched requests, for every task in the MPMD-Size-MTS instance created by the reduction, the processing cost is a monotone non-increasing function of the state size. Our algorithm will exploit this by maintaining the invariant that our state is always at least large as that of σ. The technical crux here is to show that when our state is smaller, we can augment our state in a cost-efficient manner. If the MTS states were matchings, we can consider augmenting paths in the symmetric difference of our matching and that of σ. Motivated by this, we define analogs of the symmetric difference and augmenting paths to augment our state. We believe these ideas are useful for other interesting special classes of set delay functions.

Step 3: Applying MTS algorithms. There are two issues that prevent us from applying existing MTS algorithms directly. First, the cost bounds of all known algorithms for MTS have an additive term that is equal to the diameter of the MTS state space, and the MTS instance created by our reduction has state space with diameter much larger than the optimal. The second issue is that our reduction creates an MTS instance whose state space is constructed online, i.e. the states arrive over time. At a high level, the first issue can be overcome by a guess-and-double approach. The second issue is a problem for randomised MTS algorithms that rely on embedding the state space into a tree as a pre-processing step. This is overcome by using the online embedding of [10]. See Section 4.3 for a more detailed discussion.

Designing a deterministic algorithm for MPMD with Concave Delay. We use the (offline) moat-growing framework (generally used for constrained connectivity problems) by Goemans and Williamson [22], to design an online deterministic linear programming-based algorithm for MPMD with uniform concave delay. Our algorithm is a modification of an existing linear programming-based algorithm due to Bienkowski et al. [12], who give a deterministic competitive algorithm for MPMD with uniform linear delay. Their algorithm heavily relies on the fact that, when the delay function is linear, requests accumulate delay cost at the same rate at all times, regardless of their arrival time. The main challenge in applying the framework to concave delay is that, unlike in the case of linear delay, the requests can accumulate delay at different rates at any point in time, depending on their arrival time. See the full version of the paper on ArXiv for a detailed discussion.

1.2 Related Work

MPMD was introduced by Emek et al. [19] where the delay functions associated with each request are uniform linear. They designed a randomised algorithm that achieves a competitive ratio of $O(\log^2 n + \log \Delta)$, where n is the number of points in the metric space and Δ is its aspect ratio. Azar et al. [3] used a randomised HST embedding to provide a $O(\log n)$-competitive almost-deterministic algorithm, improving Emek et al.'s bound and removing the dependency on the aspect ratio of the metric space. Furthermore, they provided a lower bound of $\Omega(\sqrt{\log n})$ for any randomised algorithm in the case of linear delay. Ashlagi et al. [1] improved this lower bound to $\Omega(\frac{\log n}{\log \log n})$ and $\Omega(\sqrt{\frac{\log n}{\log \log n}})$ for the bipartite case, which are the best known so far. Liu et al. furthermore adapted the algorithm by Azar et al. to the bipartite setting and improved the analysis of Emek et al.'s algorithm to $O(\log n)$. The next deterministic algorithm for simple metrics was by Emek et al. [20] who proved a competitive ratio of 3 for the simple metric space of 2 points. The first deterministic algorithm for general metric spaces was by Bienkowski et al. [13] and their analysis resulted in a competitive ratio of $O(m^{2.46})$. Bienkowski et al. [12] and Azar et al. [6] concurrently and independently improved this bound to $O(m)$ and $O(m^{0.59})$ respectively, introducing the first linear and sub-linear deterministic solutions to the problem. The algorithms above assumed the delay cost to be given by a uniform linear delay function associated with each individual request.

Liu et al. [27] was the first to consider convex delay functions and demonstrated an interesting gap between the solutions for the case with linear delay and convex delay on a uniform metric space by giving a deterministic asymptotically optimal $O(m)$-competitive algorithm for the uniform metric space.

Azar et al. [8] subsequently considered the problem with concave delay and achieved an $O(1)$-competitive deterministic algorithm for the single point metric space and an $O(\log n)$ randomised algorithm for general metric spaces.

The above algorithms assumed all requests incurred delay in accordance with uniform delay functions and regarded the delay function to be associated with each individual request. Furthermore, all prior solutions to MPMD assumed clairvoyance. To the best of our knowledge, no one has considered the non-clairvoyant generalisation of the problem where the delay function depends on the set of unmatched requests.

Non-clairvoyant algorithms nevertheless have been designed for other online problems such as the Set Cover problem [4, 26], the k-server problem [25], and multi-level aggregation [26].

The notion of introducing delay to online problems originated well before it was applied to online metric matching and finds applications in amongst others aggregating messages in computer networks, aggregating orders in supply-chain management, and operating systems. See [4, 5, 7, 9, 11, 14, 17, 18, 21, 23, 26, 28] for further reading. All problems above define the cost of delay as a function associated with each request. To the best of our knowledge, no online problems with delay have so far defined the cost of delay as an arbitrary function of the set of unmatched requests.

2 Preliminaries

In this section we introduce our notation and give formal definitions for set delay and size-based delay functions, as well as MPMD-Set.

2.1 Min-cost Perfect Matching with Delay

The min-cost perfect matching with delays (MPMD) problem, introduced by Emek et. al [19] is defined on a metric space (V, d), which consists of a set of points V and distance function $d : V \times V \to \mathbb{R}^+$. An online input instance over (V, d) is a sequence of requests $R = (r_1, ..., r_m)$ that arrive at points in the metric space over time. Each $r_k \in R$ has an associated position and arrival time. We assume, without loss of generality, that time is divided into discrete timesteps.

Upon the arrival of a request, the algorithm must choose to either match the request, incurring a cost equal to the distance between the two requests in the metric space, or to delay the request, incurring a cost given by a delay function associated with the request, in the hope of finding a more suitable match in the near future.

A solution produced by an online matching algorithm is a sequence of matchings $\mathbb{M} = (M_0...M_{final})$, where M_i is the matching associated with the ith timestep. Note that we assume that requests only arrive at the start of a timestep. A solution $\mathbb{M}$ must satisfy the following properties:

- $M_0 = \emptyset$
- M_{final} is a perfect matching
- For all i, $M_i \subseteq M_{i+1}$

We refer to the third property as *monotonicity*. The cost associated with a solution $\mathbb{M}$ consists of the sum of the distances between matched requests in M_{final} plus the sum of the delay costs incurred by all requests. The aim of an online matching algorithm is to produce a sequence of matchings that satisfies the above properties with minimal cost.

In the original MPMD problem, the delay cost incurred by a request is the time between its arrival and when it was matched. In MPMD-Set, the instantaneous delay incurred at a timestep t can be an arbitrary function of the set of currently unmatched requests U_t. The total delay cost is the sum over timesteps t of $f_t(U_t)$ where f_t is a set delay function as defined below.

▶ **Definition 8** (Set delay function). *Let U be a set of requests. We define a delay function $f_t : 2^U \to \mathbb{R}^{\geq 0}$, for any timestep t, to be a* set delay *function if it satisfies the following properties:*

- $f_t(\emptyset) = 0$
- $A \subseteq B \Rightarrow f_t(A) \leq f_t(B)$
- *For all $\emptyset \neq U \in 2^V$, we have $\sum_{t=0}^{\infty} f_t(U) = \infty$*

The last property implies that all requests must eventually be matched.

In MPMD-Size, the set delay function is a size-based delay function, as defined below.

▶ **Definition 9** (Size-based delay function). *We define a delay function $f_t : 2^U \to \mathbb{R}^{\geq 0}$ to be* size-based *if, for any timestep t, it satisfies all properties of a set delay function and is monotone non-decreasing as a function of the size of the set of requests U.*

3 A Lower Bound for MPMD-Set

In this section, we prove Theorem 1.

Proof of Theorem 1. Consider a four-point metric space (as depicted in figure 1) with three points at distance ϵ from one another, and the fourth point (p_4) at distance D from the other points, where D is the diameter of the metric space.

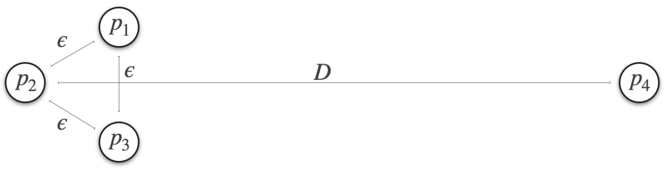

Figure 1 A visualisation of the four-point metric space.

We define a request sequence of six requests $R = (r_1, r_2, r_3, r_4, r_5, r_6)$ where the first four requests arrive at time $t = 0$ and the latter two arrive at time $t = 2$. For each $i \in \{1...4\}$, we place request r_i on point p_i. At $t = 2$, we then place request r_5 on p_3 and r_6 on p_4. In terms of delay, we are working with the special case of deadline functions. A *deadline function* is a delay function that is 0 up until some time d, called the deadline, and ∞ afterwards. When the deadline of a request is reached, the algorithm must ensure that the request is matched.

At $t = 0$, request r_1 reaches its deadline and hence the algorithm will need to match two requests. Since the algorithm is non-clairvoyant, the algorithm has no knowledge of the deadlines of future requests. We therefore assume without loss of generality that it matches r_1 to r_3 and pays a distance cost of ϵ. At $t = 1$, r_2 reaches its deadline and the algorithm is forced to match it to r_4 at a distance cost of D. At $t = 2$, the final two requests will arrive and instantly reach their deadline. The algorithm will consequently need to match r_5 to r_6 at a distance cost of D. The total cost of ALG is $2D + \epsilon$.

The optimal offline solution OPT is to match r_1 to r_2 and match locally at $t = 2$ on points p_3 and p_4. The total cost of OPT is therefore ϵ. The competitive ratio of the algorithm is $\Omega(D/\epsilon)$ and D/ϵ is the aspect ratio of the metric space, as desired. ◀

4 An Online Reduction from MPMD-Set to MTS

In this section, we prove Theorem 2 by defining a reduction from MPMD-Set to MTS. We start by translating an arbitrary instance of MPMD-Set into an instance of MTS in Section 4.1. In Section 4.2, we show that we can transform an arbitrary MPMD-Size-MTS solution into a monotone solution of the same or less cost. As observed in the Introduction, a monotone schedule directly corresponds to a solution for the online matching with delay problem of equal cost. This completes the proof of Theorem 2. We finish this section with a proof of Corollary 3.

4.1 Translating an instance of MPMD-Set into and instance of MTS

We define the set of internal states of the MTS instance to be the set of all possible subsets of the requests that have arrived so far in the MPMD-Set instance. In order to define the transition cost associated with moving between states, we define the following graph.

▶ **Definition 10** (Transition graph G). *The nodes of G are the states of the MPMD-Set-MTS instance. We define an undirected edge $(S, S') \in E$ between states S and S' if $S = S' \cup \{r, r'\}$ and define its cost $c(S, S')$ to be $d(r, r')$. Paths in the transition graph are called* transition paths.

We define the transition cost $c(S, S')$ of moving between arbitrary states S and S' in the MTS instance to be the cost of the shortest path between them in the transition graph G. Note that a path in G between states S, S' corresponds to a sequence of transitions from S to S', where each edge in the path corresponds to a transition that either adds two requests or removes two requests. Also, note that any schedule σ, produced by an online MTS algorithm, corresponds to a path in the metric completion of G and the total transition cost incurred by σ is simply the total cost of the path. Each task in the MTS instance is associated with a given timestep in the MPMD-Set problem. The cost vector associated with each task is, for every state S, the instantaneous delay cost accumulated by the set of requests not in S.

▶ **Definition 11** (R_i). *We define R_i to be the set of requests that have arrived up to and including timestep i in the original MPMD-Set instance.*

The total cost of a schedule $\sigma = (S_0, ..., S_T)$ can be expressed as follows.

$$\text{cost}(\sigma) = \sum_{i=0}^{T-1} c(S_i, S_{i+1}) + f_i(R_i \setminus S_i).$$

By construction, the cost associated with processing the tasks represent the delay cost incurred by the requests, while the distance cost is represented by the transition costs associated with moving between states.

4.2 Size-based delay functions admit monotone scheduling algorithms

In this subsection, we prove the existence of an online algorithm that converts an arbitrary MPMD-Size-MTS solution into a monotone solution, without incurring any extra cost.

▶ **Lemma 12.** *There exists an online algorithm that converts an arbitrary MPMD-Size-MTS solution into a monotone solution of the same or less cost.*

Proof. To prove Lemma 12, we define an online *state conversion algorithm* (Sensible-ALG) which, for every state S_i in a schedule σ, produced by an *arbitrary* online scheduling algorithm (OSA), produces a state S'_i such that the cost of the schedule $\sigma' = (S'_0, \ldots, S'_{|\sigma|})$ is at most the cost of the original schedule σ, and σ' is monotone.

The state conversion algorithm aims to maintain the invariant that $|S'_i| \geq |S_i|$ and the main property of a monotone schedule, which is that $S'_{i-1} \subseteq S'_i$ for every i. At a high level, it does this by adding requests to its current state when the invariant is violated that allows it to also move closer to the current state of σ. The algorithm Sensible-ALG uses the following analog of the symmetric difference of two matchings to augment its current state in a cost-efficient manner.

▶ **Definition 13** ((A, B)-difference graph). *Let H be a (multi-)graph with vertex set R, and A, B be states. The graph H is an (A, B)-*difference graph *if it is a T-join with $T = A \triangle B$: for every $r \in R$, the degree of r in H is odd if and only if $r \in A \triangle B$.*

For example, one way to obtain an (A, B)-difference graph is by taking a perfect matching M_A on A and a perfect matching M_B on B and then taking the symmetric difference of M_A and M_B.

▶ **Definition 14** (P-difference graph and realisable difference graphs)**.** *Let A, B be states and P be a transition path between A and B. The P-*difference graph $\mathrm{diff}(P)$ *is a (multi-)graph on vertex set R where the multiplicity of the edge (p, q) is equal to the number of transitions along the path P that adds or removes the $\{p, q\}$. An (A, B)-difference graph H is* realisable *if it is the P-difference graph for a transition path P between A, B.*

Note that if P is a transition path between states A, B, then a P-difference graph is an (A, B)-difference graph. Moreover, if P is a shortest path between S and S', then the total cost of the edges in $\mathrm{diff}(P)$ is exactly equal to $c(S, S')$. We also note that not all difference graphs are realisable[2]

We now characterise the structure of difference graphs that correspond to shortest paths in the transition graph.

▶ **Definition 15** (Canonical difference graphs)**.** *Let H be an (A, B)-difference graph. We say that H is* canonical *if it can be decomposed into a collection of $\ell := |A \triangle B|/2$ edge-disjoint paths $Q_1, \ldots, Q_\ell$ between disjoint pairs (p_i, q_i) of $A \triangle B$ such that for each i, Q_i consists of a single edge (p_i, q_i) if either both p, q are in $A \setminus B$ or both in $B \setminus A$, and Q_i consists of two edges $(p, s), (s, q)$ for some other request s if exactly one of p, q is in $A \setminus B$ and the other in $B \setminus A$.*

▶ **Proposition 16.** *If H is a canonical (A, B)-difference graph, then H is realisable.*

Proof. We show by induction on $\ell = |A \triangle B|/2$ that there is a transition path P from A to B. Consider the base case $\ell = 1$. Suppose Q_1 consists of a single edge (p, q). If $p, q \in A \setminus B$, then we can transition directly from A to B by removing p, q; otherwise, we add p, q. Next, suppose that Q_1 consists of two edges $(p, s), (s, q)$. Consider the case that $p \in A \setminus B$ and $q \in B \setminus A$. If $s \in A \cap B$, then we can transition from A to B by first removing p, s and then adding s, q; otherwise we first add s, q and then remove p, s. The case when $q \in A \setminus B$ and $p \in B \setminus A$ follows similarly. In all of these cases, we have that H is exactly the P-difference graph where P is the transition path corresponding to the sequence of transitions used.

Suppose the statement is true for ℓ up to $k - 1$ and let H be a canonical difference graph with k edge-disjoint paths. Let A' be the state after executing the above procedure on Q_1 and P_1 be the transition path corresponding to the sequence of transitions from A to A'. As argued above, Q_1 is the P_1-difference graph. Since $H \setminus Q_1$ is a canonical (A', B)-difference graph with $k - 1$ edge-disjoint paths, we can apply induction to get a transition path P_2 from A' to B such that $H \setminus Q_1$ is the P_2-difference graph. By concatenating P_1 and P_2, we get a transition path P from A to B. Moreover, H is the P-difference graph, as desired. ◀

▶ **Lemma 17.** *Let A, B be states. There exists a shortest path P in the transition graph between A, B such that* $\mathrm{diff}(P)$ *is canonical.*

Proof. Since $\mathrm{diff}(P)$ is an $(A \triangle B)$-join, it contains a collection of $\ell := |A \triangle B|/2$ edge-disjoint paths $Q_1, \ldots, Q_\ell$ connecting disjoint pairs of requests $p_i, q_i \in A \triangle B$.

We now shortcut these paths to produce a canonical (A, B)-difference graph H. In particular, for each i, if $p_i, q_i \in A \setminus B$ or $p_i, q_i \in B \setminus A$, add the edge (p_i, q_i) to H; otherwise, $p_i \in A \setminus B$ and $q_i \in B \setminus A$ (or vice versa), there must be an intermediate node s_i on the path

[2] For example, consider the difference graph H consisting of requests p, q, r and edges (p, q) and (q, r). Observe that H is an $(\{p, r\}, \emptyset)$-difference graph but there is no transition path P from $\{p, r\}$ to $\emptyset$ such that $\mathrm{diff}(P) = H$.

Q_i and we add the edges $(p_i, s_i), (s_i, q_i)$. The existence of s_i is because if Q_i only consists of a single edge (p_i, q_i) then p_i and q_i must be both in $A \setminus B$ or $B \setminus A$, otherwise the transition in P that adds or removes p_i, q_i is invalid.

By triangle inequality, the total cost of the edges in H is at most that of $Q_1 \cup \cdots \cup Q_\ell$ which in turn is contained in $\mathrm{diff}(P)$. Since H is a canonical (A, B)-difference graph, there exists a transition path P' with $\mathrm{diff}(P') = H$. Since $c(P')$ is equal to the total cost of the edges in $\mathrm{diff}(P') = H$, we get the lemma. ◀

▶ **Proposition 18.** *Let A, B be states such that $|A| = |B| + 2$. Let P be a shortest path in the transition graph between A, B such that $\mathrm{diff}(P)$ is canonical and $Q_1, \dots, Q_\ell$ be the corresponding collection of $|A \triangle B|/2$ edge-disjoint paths connecting disjoint pairs of $A \triangle B$. Then, one such path Q_i is a single edge (p, q) with $p, q \in A \setminus B$.*

This proposition follows from the fact that $|A \setminus B| = |B \setminus A| + 2$ and so there must be a path Q_i connecting $p, q \in A \setminus B$. By definition of canonical difference graphs, Q_i is a single edge (p, q).

We are now ready to formally define Sensible-ALG.

Description of Sensible-ALG. Our online algorithm takes as input a sequence of states $\sigma = (S_1, \dots, S_{|\sigma|})$ produced by an online scheduling algorithm. We assume that σ satisfies that for all $0 \le i < |\sigma| - 1$, S_i and S_{i+1} are neighbours in G. This is without loss of generality: if the input schedule does not satisfy these properties then we can add intermediate timesteps and all states on the shortest path between S_i and S_{i+1} such that it satisfies the above property, and the cost remains the same. When a new state S_i arrives, if $|S_i| \le |S'_{i-1}|$, the algorithm sets $S'_i = S'_{i-1}$; otherwise, it computes a shortest transition path P between S_i and S'_{i-1} such that $\mathrm{diff}(P)$ is canonical, and sets $S'_i = S'_{i-1} \cup \{p, q\}$ where $p, q \in S_i \setminus S'_{i-1}$ and (p, q) is a path in the path decomposition of $\mathrm{diff}(P)$.

Since Sensible-ALG always transitions to a state that is a superset of its previous state, its solution σ' is monotone.

Next, we analyse the transition cost of σ'.

▶ **Lemma 19.** $\sum_{i=0}^{|\sigma|-1} c(S'_{i-1}, S'_i) \le \sum_{i=0}^{|\sigma|-1} c(S_{i-1}, S_i)$.

Proof. We prove this lemma by introducing the potential $\phi_i = c(S_i, S'_i)$.

We claim that in each iteration i, the transition cost of σ' is at most the transition cost of σ plus the decrease in the potential.

▷ Claim 20. For all i, we have $c(S'_{i-1}, S'_i) \le c(S_{i-1}, S_i) - (\phi_i - \phi_{i-1})$.

Proof of Claim 20. By triangle inequality, we have

$$c(S_i, S'_{i-1}) \le c(S_{i-1}, S_i) + c(S_{i-1}, S'_{i-1}). \tag{1}$$

Next, we will show that

$$c(S'_{i-1}, S'_i) \le c(S_i, S'_{i-1}) - c(S_i, S'_i). \tag{2}$$

Combining these two inequalities yields the claim.

Observe that Inequality (2) holds when $S'_i = S'_{i-1}$. Suppose $S'_i \ne S'_{i-1}$. In this case, the algorithm computes the shortest path P between S_i and S'_{i-1} that is canonical and set $S'_i = S'_{i-1} \cup \{p, q\}$ where $p, q \in S_i \setminus S'_{i-1}$ and (p, q) is a path in the path decomposition of $\mathrm{diff}(P)$. We have that $\mathrm{diff}(P) \setminus (p, q)$ is a canonical (S_i, S'_i)-difference graph and thus, by

Proposition 16, corresponds to a transition path between S_i, S'_i with length $c(S_i, S'_{i-1}) - c(p,q) = c(S_i, S'_{i-1}) - c(S'_{i-1}, S'_i)$. Thus, $c(S_i, S'_i) \leq c(S_i, S'_{i-1}) - c(S'_{i-1}, S'_i)$. Rearranging this inequality yields Inequality (2). This completes the proof of the claim. $\lhd$

Using Claim 20, we determine the total transition cost incurred by σ' as follows.

$$\begin{aligned}\sum_{i=1}^{|\sigma|} c(S'_{i-1}, S'_i) &\leq \sum_{i=1}^{|\sigma|} c(S_{i-1}, S_i) - \sum_{i=1}^{|\sigma|} (\phi_i - \phi_{i-1}) \\ &= \sum_{i=1}^{|\sigma|} c(S_{i-1}, S_i) - \phi_{|\sigma|} + \phi_0 \\ &\leq \sum_{i=1}^{|\sigma|} c(S_{i-1}, S_i).\end{aligned}$$

The last inequality holds because $\phi_0 = 0$ and the potential is non-negative. ◀

The cost of an MPMD-Set-MTS solution consists of the transitions cost, as well as the processing cost. Since in every iteration i, we have $|S'_i| \geq |S_i|$ and the processing cost of a state is a monotone non-increasing function of the size of the state, we get that the total processing cost of σ' is at most that of σ. Together with Lemma 19, we get that the total cost of σ' is at most that of σ. This concludes the proof of Lemma 12. ◀

4.3 Applying MTS Algorithms to MPMD-Set-MTS

In this section, we prove Corollary 3. Consider an instance of MPMD-Set with m requests in a metric space of n points and the instance of MPMD-Set-MTS created by applying Theorem 2. Let N be the number of states of the MPMD-Set-MTS instance.

There are two issues that arise when applying MTS algorithms to MPMD-Set-MTS directly.

4.3.1 Eliminating the Diameter

The first issue is that all known MTS algorithms have a cost bound of the form $f(N) \cdot cost(OPT) + D$ where OPT is the optimal MTS solution and D is the diameter of the MTS state space. Observe that D is at least the distance between the empty matching and the max-cost perfect matching, i.e. the cost of the max-cost perfect matching. Unfortunately, the cost of the max-cost perfect matching can be much larger than that of the optimal solution. To overcome this, one could restrict the MTS solution to only use states whose distance from the initial state is at most $cost(OPT)$. This can be achieved by setting the costs of the other states to be infinite. This effectively reduces the diameter of the state space to at most $2 \cdot cost(OPT)$, and would give us a cost bound of $O(f(N)) \cdot cost(OPT) + 2 \cdot cost(OPT)$. The issue is that, since the MTS tasks arrive in an online fashion, the optimal solution remains unknown until all tasks have arrived. To address this issue we use the Guess-and-Double Method, which, maintains a guess of the value of the optimal solution as the tasks are processed. This guess is used to determine the diameter of the state space used by the algorithm. When the guess becomes too small, the value of the guess is increased and the algorithm is simulated on the input that has already been processed, only this time on the larger state space, to determine the state it would now be in, and processes the new tasks accordingly. For the benefit of the reader, we give a high level overview of the method below.

The Guess-and-Double Method. Let $R = [r_1, r_2, ...r_T]$ be the sequence of tasks given by the MTS problem. Let OPT_t be the cost of the optimal solution for processing the tasks that have arrived up to and including time t. At any time t we maintain a *guess* j such that

$$2^{j-1} < OPT_t \leq 2^j. \tag{3}$$

When our guess j no longer satisfies (3) we increase it's value (thus doubling the radius of the metric space) until it is back within our bounds. Each new guess instantiates a new *phase*. At the start of each phase, we *restart* the MTS algorithm with a superset of the previous state space, which now consists of all states within distance 2^j from the original start state. Note that by *restart* we mean that we simulate the MTS algorithm with the new diameter on all tasks that have arrived so far, and process the new tasks in accordance with the decisions made by the algorithm with the new diameter.

Our initial guess j satisfies $2^{j-1} < OPT_1 \leq 2^j$. We define MTS_j to be the MTS algorithm that operates on the given state space with diameter 2^j. Let $R_t = [r_1, ..., r_t]$ be the sequence of tasks that have arrived up to and including timestep t. We define $MTS_j(R_t)$ to be the state MTS_j would end up in after processing R_t. The Guess-and-Double method, for every timestep t, maintains a guess j that satisfies (3), and moves to $MTS_j(R_t)$.

Algorithm 1 Updating the guess and splitting the tasks into phases.

```
1  Initialise the first phase.
2  Choose $j$ such that $2^{j-1} < OPT_1 \leq 2^j$.
3  for Every timestep $t$ do
4      if $2^{j-1} < OPT_t \leq 2^j$ then
5          Move to state $MTS_j(R_t)$
6      end
7      else
8          End the previous phase and initialise a new phase.
9          Update the value of $j$ such that $2^{j-1} < OPT_t \leq 2^j$.
10         Move to state $MTS_j(R_t)$
11     end
12 end
```

Note that each time we update the value of our guess, the value of the optimal solution has at least doubled since we made our last guess.

To analyse the total cost of the resulting schedule we look at two separate costs. The first is the cost incurred within each phase. This includes the transition costs incurred during the phase (from moving between states), plus the cost associated with servicing all tasks that arrived during the phase in the states the algorithm moved through during the phase. We refer to this cost as the *internal* phase cost. The second cost consist of the transition costs incurred in moving between the last state of a phase i, and the first state of the consecutive phase $i + 1$. We refer to this cost as the *external* phase cost.

We start by bounding the internal phase cost. Let $cost(MTS_j)$ denote the internal phase cost of the phase associated with guess j. Because $cost(MTS_j)$ can be upper bounded by the cost the algorithm would have incurred for processing all tasks that arrived prior to and during the phase associated with guess j, we can bound $cost(MTS_j)$ as follows:

$$cost(MTS_j) \leq (O(f(N)) + 2) \cdot cost(OPT_j).$$

Therefore, the total internal phase cost incurred over all k phases is

$$\begin{aligned}\sum_{i=1}^{k} ALG_i &\leq (O(f(N)) + 2) \cdot \sum_{i=1}^{k} cost(OPT_i) \\ &\leq 2 \cdot (O(f(N)) + 2) \cdot cost(OPT_k).\end{aligned}$$

Next, we bound the external phase cost. Let S_i be the last state of a given phase i, associated with a guess j, and let S'_{i+1} be the first state of phase $i+1$, associated with the next guess j'. We bound $c(S_i, S'_{i+1})$[3] as follows:

Let S_0 be the empty start state.

$$c(S_i, S'_{i+1}) \leq c(S_i, S_0) + c(S'_{i+1}, S_0).$$

For all consecutive phases i and $i+1$, associated with guesses j and j' respectively, it holds that

$$c(S_i, S_0) \leq (O(f(N)) + 2) \cdot cost(OPT_i)$$

and

$$c(S'_{i+1}, S_0) \leq (O(f(N)) + 2) \cdot cost(OPT_{i+1}).$$

We thus bound the cost over all $k-1$ phase transitions as follows

$$\begin{aligned}\sum_{i=1}^{k-1} c(S_i, S'_{i+1}) &\leq 2 \cdot \sum_{i=1}^{k-1} (O(f(N)) + 2) \cdot cost(OPT_{i+1}) \\ &\leq 4 \cdot (O(f(N)) + 2) \cdot cost(OPT_k).\end{aligned}$$

The total cost of the solution produced by the Guess-and-Double method can thus be bounded by $6 \cdot (O(f(N)) + 2) \cdot cost(OPT_k)$.

We can now use the $O(N)$-competitive deterministic algorithm of [15] to obtain our deterministic algorithm for MPMD-Size.

4.3.2 The Need for an Online Embedding

The second issue stems from the fact that the reduction in Theorem 2 creates an MTS instance where the states are arriving over time. This is because the states correspond to matchings of requests and the requests are arriving online. This does not pose a problem for the deterministic $O(N)$-competitive Work Function Algorithm of [15]. However, we cannot directly apply the current-best randomised algorithm for MTS of [16] as it pre-computes a probabilistic embedding of the MTS metric space into a hierarchically separated tree (HST). Instead, we need to use a probabilistic online embedding into a HST together with the $O(\log N)$-competitive randomized algorithm for MTS on HSTs of [16]. Using the online embedding of [24] adds a factor of $O(\log N \log \Phi)$ where Φ is the ratio of the largest distance to the smallest distance in the MTS state space, i.e. the aspect ratio. However, Φ can be arbitrarily large. We deal with this by proving that the Abstract Network Design Framework of [10] can be extended to apply to MTS. For the benefit of the reader, we give a short overview of the Abstract Network Design Framework.

[3] Recall that we denote the transition cost of going from state S_i to S'_{i+1} by $c(S_i, S'_{i+1})$.

The Abstract Network Design Framework. In an instance of abstract network design, the algorithm is given a connected graph $G(V, E)$ with edge lengths $d : E \to \mathbb{R}_+$. At each timestep i, a requests that consists of a set of points in the graph, called *terminals*, arrive in an online fashion. The algorithm provides a response $R_i = (G_i, C_i)$, which consists of a subgraph $G_i \subseteq E$, and a connectivity list C_i, which is an ordered subset of terminal pairs from the terminals that have arrived so far, and determines what will become (and remain) connected from this timestep onwards. The algorithm is given, at each timestep, a feasibility function $F_i : (C_1, ..., C_i) \to \{0, 1\}$ that maps a sequence of connectivity lists to either 0 (infeasible) or 1 (feasible). A solution S_i, which consists of a sequence of responses for every time step up to and including timestep i, is feasible if $F_i(C_1, ..., C_i) = 1$ and all pairs in C_j are connected in all G_i for all $i \geq j$. To determine the cost of a solution to the first i requests S_i, the framework uses a load function $\rho_i : 2^{\{1,\ldots,i\}} \to \mathbb{R}_+$, which takes as input the sequence of timesteps in which the edge was used, and outputs a corresponding multiplier to the cost of an edge $d(e)$. It aims to model how the cost of using an edge grows as the edge is used multiple times. The function must be subadditive, monotone non-decreasing, and satisfy $\rho_i(I) = 0$ if and only if $I = \emptyset$. The total cost of a solution S_i is defined as follows.

$$\mathit{cost}(S_i) = \sum_{e \in E} d(e) \cdot \rho(\{j \leq i : e \in G_j\})$$

Extending the Abstract Network Design Framework. Though we can express the transition cost of an instance of general MTS using this framework, we cannot express the cost vectors associated with processing the tasks in MTS in the current state of the framework. In order to address this issue, we propose the following alterations to generalise the framework.

We replace the feasibility function with a function $F'_i : (C_1, ..., C_i) \to \mathbb{R}_+$, where $F'_i(C_1, ..., C_i)$ is the processing cost of the algorithm during timestep i if the solution $S_i = ((G_1, C_1), \ldots, (G_i, C_i))$ is feasible, and ∞ otherwise.

We now re-define the cost of a solution to the first i requests S_i to incorporate the total processing cost incurred by the algorithm.

$$\mathit{cost}(S_i) = \sum_{e \in E} d(e) \cdot \rho(\{j \leq i : e \in G_j\} + \sum_{l=1}^{i} F'(C_1, \ldots, C_l)$$

Since the processing cost provided to the algorithm is independent of the metric space, it follows that the processing cost remains unaffected by the online embedding. It therefore does not affect the overhead due to the online embedding.

Note that the extension of this framework means it can now be used to model online problems with delay, where the delay cost can be modelled as the processing cost.

Expressing MTS in the Abstract Network Design Framework. It remains to formulate the general MTS problem in the Extended Abstract Network Design Framework defined above.

We define the terminal set of the ith request to be the set of states that have arrived so far. We define the cost of an edge $d((u, v))$ to be the transition cost between states u, v. Let $\mathbb{T}_i$ be the cost vector associated with processing task i, and $T_i(w)$ be the cost of processing task i in state w. Let v be the last terminal in C_i. The extended feasibility function F_i is defined by $F_i(C_1, \ldots, C_i) = T_i(v)$ if there is only a single ordered pair in each C_j for all $j \leq i$ and the sequence of $(C_1, \ldots, C_i)$ is a valid path. The load function is simply the cardinality function because we pay the transition cost associated with the edge each time we transition to a different state.

Min-operator. Bartal et al. [10] show that if the problem can be captured by the Abstract Network Design Framework and admits a *min-operator*, it is possible to reduce the overhead due to the online embedding to $O(\log^3 N)$.

▶ **Definition 21** (Min-operator)**.** *An algorithm admits a min-operator with factor* $\mu \geq 1$ *if there exists a competitive algorithm*[4] *for the problem, and for any two deterministic online algorithms* A *and* B[5]*, there exists a third online deterministic algorithm* C *such that the cost of* C *satisfies* $cost(C) \leq \mu \cdot \min\{cost(A), cost(B)\}$*, where* $cost(A)$ *and* $cost(B)$ *are the respective costs of algorithms* A *and* B*. If either algorithms* A *or* B *are randomised, the expected cost of* C *must satisfy* $E[cost(C)] \leq \mu \cdot \min\{E[cost(A)], E[cost(B)]\}$*.*

We have shown that the results by Bartal et al. [10] also hold for the extended Abstract Network Design Framework. Since the MTS problem in general admits a min-operator [2], using the framework of [10] allows us to reduce the overhead due to the online embedding to $O(\log^3 N)$ for an overall competitive ratio of $O(\log^4 N)$.

References

1 Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017*, volume 81 of *LIPIcs*, pages 1:1–1:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.1.

2 Yossi Azar, Andrei Z. Broder, and Mark S. Manasse. On-line choice of on-line algorithms. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms*, pages 432–440. ACM/SIAM, 1993. URL: http://dl.acm.org/citation.cfm?id=313559.313847.

3 Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 1051–1061. SIAM, 2017. doi:10.1137/1.9781611974782.67.

4 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*, volume 173 of *LIPIcs*, pages 8:1–8:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

5 Yossi Azar, Yuval Emek, Rob van Stee, and Danny Vainstein. The price of clustering in bin-packing with applications to bin-packingwith delays. In *Proceedings of the 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019*, pages 1–10. ACM, 2019. doi:10.1145/3323165.3323180.

6 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.

7 Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. *ACM Trans. Algorithms*, 17(3):23:1–23:31, 2021. doi:10.1145/3459925.

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

[4] This algorithm may be randomised.
[5] Note that these algorithms need not be competitive on all instances of the problem.

9 Yossi Azar and Noam Touitou. Beyond tree embeddings – A deterministic framework for network design with deadlines or delay. In Sandy Irani, editor, *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 1368–1379. IEEE, 2020. doi:10.1109/FOCS46700.2020.00129.

10 Yair Bartal, Nova Fandina, and Seeun William Umboh. Online probabilistic metric embedding: A general framework for bypassing inherent bounds. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 1538–1557. SIAM, 2020. doi:10.1137/1.9781611975994.95.

11 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jirí Sgall, Kim Thang Nguyen, and Pavel Veselý. Online algorithms for multilevel aggregation. *Oper. Res.*, 68(1):214–232, 2020. doi:10.1287/opre.2019.1847.

12 Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Pawel Schmidt. A primal-dual online deterministic algorithm for matching with delays. In *Approximation and Online Algorithms – Proceedings of the 16th International Workshop, WAOA 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, volume 11312 of *Lecture Notes in Computer Science*, pages 51–68. Springer, 2018. doi:10.1007/978-3-030-04693-4_4.

13 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. A match in time saves nine: Deterministic online matching with delays. In *Approximation and Online Algorithms – Proceedings of the 15th International Workshop, WAOA 2017*, volume 10787 of *Lecture Notes in Computer Science*, pages 132–146. Springer, 2017. doi:10.1007/978-3-319-89441-6_11.

14 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. Online service with delay on a line. In Zvi Lotker and Boaz Patt-Shamir, editors, *Structural Information and Communication Complexity – Proceedings of the 25th International Colloquium, SIROCCO 2018, Ma'ale HaHamisha, Israel, June 18-21, 2018, Revised Selected Papers*, volume 11085 of *Lecture Notes in Computer Science*, pages 237–248. Springer, 2018. doi:10.1007/978-3-030-01325-7_22.

15 Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992. doi:10.1145/146585.146588.

16 Sébastien Bubeck, Michael B. Cohen, James R. Lee, and Yin Tat Lee. Metrical task systems on trees via mirror descent and unfair gluing. *SIAM J. Comput.*, 50(3):909–923, 2021. doi:10.1137/19M1237879.

17 Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 1235–1244. SIAM, 2017. doi:10.1137/1.9781611974782.80.

18 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*, volume 229 of *LIPIcs*, pages 40:1–40:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

19 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 333–344. ACM, 2016. doi:10.1145/2897518.2897557.

20 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.

21 Leah Epstein. On bin packing with clustering and bin packing with delays. *Discret. Optim.*, 41:100647, 2021. doi:10.1016/j.disopt.2021.100647.

22 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995. doi:10.1137/S0097539793242618.

23 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1125–1138. ACM, 2020. doi:10.1145/3357713.3384277.

24 Piotr Indyk, Avner Magen, Anastasios Sidiropoulos, and Anastasios Zouzias. Online embeddings. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and proceedings of the 14th International Workshop, RANDOM 2010*, volume 6302 of *Lecture Notes in Computer Science*, pages 246–259. Springer, 2010.

25 Predrag Krnetic, Darya Melnyk, Yuyi Wang, and Roger Wattenhofer. The k-server problem with delays on the uniform metric space. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 61:1–61:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ISAAC.2020.61.

26 Ngoc Mai Le, Seeun William Umboh, and Ningyuan Xie. The power of clairvoyance for multi-level aggregation and set cover with delay. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2023. To appear.

27 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*, volume 123 of *LIPIcs*, pages 62:1–62:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.62.

28 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*, volume 212 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.53.